UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE GRAU

Grau en Enginyeria Electrònica Industrial i Automàtica

ROS I EL CÀLCUL DE TRAJECTÒRIES PER A UN ROBOT AUTÒNOM DE COMPETICIÓ

Memòria i Annexos

Autor: Marc Gómez Lerín
Director: Marc Lluva Serra

Convocatòria: Octubre 2019

Resum

L'aplicació de ROS (*Robot Operating System*) està cada vegada més present tant en el món laboral com en el competitiu. Amb aquest treball es pretén introduir al lector en aquest aspecte, portant a terme una aplicació basada en ROS dins l'àmbit d'una competició de robòtica de nivell Europeu (Eurobot).

PUCRA, l'associació de robòtica d'estudiants de l'EEBE, ha sigut des d'on ha partit aquest projecte. En cert punt aquest ha pres una altre direcció respecte a la marcada des de l'associació. Així doncs l'adaptabilitat d'un per canvis en aquest sentit ha sigut clau per prosseguir endavant.

Partint de la base d'un *package* existent, l'objectiu es modificar-lo per adaptar-lo als requeriments desitjats en condició del reglament de la competició, com a objecte d'estudi el càlcul de trajectòries. Mitjançant un protocol de proves es porta un registre dels canvis realitzats pas a pas fins a arribar a la solució final. La qual pren en principal importància l'optimització dels moviments del robot dins l'espai de la competició, concloent amb el fet de que Dijkstra resulta ser l'algoritme, tot i que el més antic, el més eficaç donades les prestacions que ROS atorga per defecte i donats els resultats de les proves realitzades.

Resumen

La aplicación de ROS (*Robot Operating System*) está cada vez más presente tanto en el mundo laboral como en el competitivo. Con este trabajo se pretende introducir al lector en este aspecto, llevando a cabo una aplicación basada en ROS dentro del ámbito de una competición de robótica de nivel Europeo (Eurobot).

PUCRA, la asociación de robótica de estudiantes de la EEBE, ha sido desde donde ha partido este proyecto. En cierto punto éste ha tomado otra dirección respecto a la marcada desde la asociación. Entonces la adaptabilidad de uno por cambios en este sentido ha sido clave para proseguir adelante.

Partiendo de la base de un *package* existente, el objetivo es modificarlo y adaptarlo a los requerimientos deseados en condición del reglamento de la competición, como objeto de estudio el cálculo de trayectorias. Mediante un protocolo de pruebas se lleva un registro de los cambios realizados paso a paso hasta llegar a la solución final. La cual toma en principal importancia la optimización de los movimientos del robot dentro del espació de la competición, concluyendo con el hecho de que Dijkstra resulta ser el algoritmo, aunque el más antiguo, el más eficaz dadas las prestaciones que ROS otorga por defecto y dado los resultados de las pruebas realizadas.

Abstract

ROS application (Robot Operating System) is increasingly present in both world of work and competitive world. With this paper is pretended to introduce the lecturer in this aspect, carrying out a ROS based application within a robotics competition of European level (Eurobot).

PUCRA, the robotics association of students of EEBE, has been where the project has begun. However, at some point that project has taken another direction regarding to the one chosen by the association. Therefor the one's adaptability for this kind of changes has been the key to keep on.

On the basis of an existing package, the objective is to modify it and adapt it to the desired requirements conditioned by the competition's regulation, being path planning as subject matter. Via a test protocol is kept a record of all the changes made step by step until the final solution is reached. Which takes on that the most important is the optimization of the robot's movements on the competition field, concluding that is a fact that Dijkstra results in being the older but the most effective algorithm, given the benefits that ROS grant by default and given the results of the tests.

Glossari

Package. Són un conjunt d'arxius que entre tots ells formen una aplicació o funció de ROS.

Gazebo. Es tracte d'una interfície típica de ROS que permet visualitzar models 3D així com realitzar simulacions amb aquests gràcies al seu motor de físiques.

Rviz. Es tracte d'una interfície típica de ROS semblant a Gazebo però amb moltes més funcionalitats, referents al tòpics.

Dijkstra. És el nom d'un famós científic en computació, però normalment aquest nom es fa servir per l'algoritme de càlcul de trajectòries que ell mateix va crear i porta el seu nom.

Gmapping. Es refereix a un procés que comporta el *mapping* o escaneig d'un espai i la localització al mateix temps. És a més un dels *packages* de ROS que permet, a partir d'un món virtual i un model 3D de robot que incorpori els sensors adequats, poder realitzar aquest procés utilitzant Gazebo i Rviz.

AMCL. De les sigles en anglès *Adaptive Monte Carlo Localization* es tracte d'un *package* de ROS basat en l'algoritme de *Monte Carlo* el qual localitza la posició d'un objecte en un espai a partir de suposicions.

Clúster de càlcul heterogeni. Es refereix a un conjunt de processadors que treballen a l'uníson com si fossin una única unitat de computació, cada un dels processadors pot ser del mateix tipus que el principal o simplement ser un coprocessador realitzant petites tasques com aritmètica de coma flotant per exemple.

Quaternió. En l'àmbit de la robòtica, és un format que representa la rotació i orientació amb un vector de 4 dimensions (x, y, z, w). Aquest format és fet servir per evitar el *gimbal lock*, un problema present en els angles d'Euler que dificulta la lectura dels angles en l'eix Y a l'hora d'apropar-se als ±90 graus.

***Message*.** És un tipus de dada que ROS utilitza i està formada per un o més d'un tipus de dades ordinàries, com poden ser *integers*, *booleans*, vectors, quaternions, etc.

Topic. Element de ROS utilitzat per nombrar el tipus d'informació que s'envia des d'un *node*. Es podria dir que un *topic* és el nom que se li dona a un *message*.

Node. Element de ROS que podria representar un esclau en un sistema de *Master-Slave*. En realitat és un programa executable des d'un codi font generat amb Python o C++, el qual generalment gestiona i processa *topics* que provenen del mateix *node* o s'envien a d'altres.

Holonòmic. És un tipus de moviment característic en els robots que incorporen certes rodes, es defineix la mobilitat d'un robot amb rodes per ser o no holonòmic. En definitiva un robot amb aquesta característica és capaç de realitzar moviments laterals i diagonals a banda dels convencionals, és a dir endavant, enrere i girar sobre l'eix central del robot.

Índex

1. Prefaci

1.1. Origen del treball

L'associació PUCRA és on s'origina el treball, la qual va incentivar l'ús de ROS per a la competició d'Eurobot. En l'edició anterior, la primera de l'associació en aquesta competició, es va realitzar un robot que simplement seguia un conjunt d'iteracions en bucle. Per aquesta edició 2019 el grup s'assabentà de la utilització de ROS per part d'altres equips així que PUCRA va decidir integrar aquest firmware al robot.

Realitzant una petita investigació sobre aquest sistema es dividí la feina en diferents parts, hom va haver de realitzar la que ara es presenta en aquesta memòria.

1.2. Motivació

La robòtica és un dels camps de l'electrònica que crida més l'atenció, el fet de formar part d'una associació on s'hi treballi al respecte és una gran oportunitat d'avanç. Eurobot per la seva part és una de les competicions de robòtica més reconeguda on hi participen països d'arreu d'Europa. Tot plegat, juntament amb el projecte que es presentava basat en ROS ajuda a entendre la motivació d'aquest treball. La qual es basa principalment, darrere d'aquests preceptes, en l'aprenentatge d'un sistema clau pensat per la robòtica d'avui en dia.

1.3. Requeriments previs i matriu de riscos

És òptim que per entendre el conjunt que forma ROS es tinguin certs coneixements previs, com l'ús de Linux, C++, Python i XML. Per una banda els llenguatges de programació més importants són objecte d'estudi en el grau cursat, tot i així tant XML com Linux han d'estar dins dels recursos a l'abast per poder entendre tot el que envolta ROS.

Una altre qüestió són les eines a utilitzar, Gazebo demanda una capacitat mínima de processament gràfic per funcionar en condicions, de tal manera que no és útil qualsevol ordinador. A més a més és

imprescindible tenir instal·lat Linux en la versió corresponent a la que es demani des de ROS segons la distribució que es vulgui fer servir.

Aquests són el requeriments bàsics per començar, en tant que com en tot projecte existeixen riscos:

- R1 Inversió insuficient. En un inici es proposa un inversió pel projecte tenint en compte el material necessari, l'associació compte amb dos equips de competició és per això que existeix la possibilitat de que es requereixin més fons dels invertits per qualsevol eventualitat que es produeixi. Per altre banda fora de l'associació no es planteja cap cost doncs ja des d'un principi es planteja com un projecte que es porta a terme dins de l'associació. Risc Molt Alt (12 Probable - Catastròfic)
- R2 Falta de material. El pressupost es limitat i hi ha material que es comparteix amb la resta d'integrants de l'associació, es porten a més dues competicions al mateix temps, per aquest fet es pot donar el cas de necessitar un material imprescindible i que no estigui disponible. Risc Alt (9 Probable - Crític)
- R3 Material inadequat. Un cop es compra el material que es necessita es pot donar el cas que resulti no ser l'adequat pel projecte, com es diu el pressupost es limitat i en cas de succeir aquesta eventualitat pot provocar un canvi en el projecte. Risc Moderat (6 Ocasional - Crític)
- R4 Personal insuficient. Segons la grandària del projecte es requereix un nombre suficient de personal qualificat per complir els objectius establerts, l'associació compte amb un nombre raonable d'integrants. Tot i així pot ocórrer la necessitat de més personal si no s'aprovisionen bé els equips de manera adequada. Risc Moderat (4 Ocasional - Sever)
- R5 Dependència d'altres persones. El projecte es divideix en tres departaments (mecànica, electrònica i programació) cada un d'ells té els seus objectius a complir ara bé requereix del compliment d'altres objectius d'altres departaments per seguir endavant, és un risc el fet de tenir personal a la vigília d'altres per poder seguir amb la seva feina, doncs escurça el temps disponible per finalitzar el projecte. Risc Molt Alt (12 Freqüent - Crític)
- R6 Corrupció de dades. La importància d'aquest treball recau en les dades (programes, arxius, dissenys, càlculs, etc.) que es generin tant per traspassar-les al robot i que així aquest pugui funcionar correctament, com les necessàries per construir-lo. Qualsevol eventualitat inesperada però possible pot causar un corrupció de les dades o la pèrdua d'aquestes. Risc Baix (3 Improbable - Crític)

Probabilitat	Conseqüències	Grau	
1 Improbable	1 Mínimes	1	Baix
	2 Severes	2	Baix
	3 Crítiques	3	Baix
	4 Catastròfiques	4	Moderat
2 Ocasional	1 Mínimes	2	Baix
	2 Severes	4	Moderat
	3 Crítiques	6	Moderat
	4 Catastròfiques	8	Alt
3 Probable	1 Mínimes	3	Baix
	2 Severes	6	Moderat
	3 Crítiques	9	Alt
	4 Catastròfiques	12	Molt Alt
4 Freqüent	1 Mínimes	4	Moderat
	2 Severes	8	Alt
	3 Crítiques	12	Molt Alt
	4 Catastròfiques	16	Molt Alt

Taula 1.1 Taula que relaciona justificadament el grau de risc amb les probabilitats i conseqüències d'aquest. La probabilitat es multiplica pel valor de la conseqüència i en resulta el valor del grau. (Font pròpia)

		PROBABILITAT			
		Freqüent	Probable	Ocasional	Improbable
CONSEQÜÈNCIES	Catastròfiques	Molt Alt	Molt Alt	Alt	Moderat
	Crítiques	Molt Alt	Alt	Moderat	Baix
	Severes	Alt	Moderat	Moderat	Baix
	Mínimes	Moderat	Baix	Baix	Baix

Taula 1.2 Matriu de riscos. (Font pròpia)

Aquests riscos prèviament numerats i classificats s'han de tenir en compte a l'hora de realitzar el projecte és per això que es genera una avaluació dels tres més problemàtics, es fa un plantejament de cada un d'ells per tal de poder mitigar-los abans de que causin un problema.

- R1. Degut a les condicions de l'associació, el pressupost ve determinat per les inversions dels patrocinadors i de la mateixa universitat, són uns fons limitats que s'han de gestionar de manera adequada, doncs s'han de compartir entre els dos equips que formen el grup d'estudiants. Per tal de no esgotar els recursos financers del projecte es necessari delimitar: quin material es prescindible i quin no, si es poden reutilitzar materials de projectes anteriors, apostar pels recursos més efectius i de menys cost possible, proposar un marge monetari dins del propi pressupost per assegurar tenir prou fons en cas de produir-se una problemàtica inesperada. Fora de l'associació no es disposa de material adequat per portar a terme el projecte per tant es crucial que aquest romangui dins de l'associació, en tot cas s'ha de preveure aportar un cost addicional al projecte si succeeix que aquest es dona per tancat dins de l'associació però no sigui acabat.
- R5. Inicialment es proposen uns objectius per cada departament i persona de tal manera que la feina queda repartida, de totes maneres existeix la possibilitat de que una part de projecte quedi encallada a causa de que una altre continuï realitzant la feina acordada, d'aquesta manera es malgasta el temps amb esperes. Per evitar que aquests temps morts causin un problema és essencial que es marquin uns *deadlines* per a cada tasca marcant el temps total que se li hauria de dedicar, en cas d'incompliment del temps acordat els companys han de prestar la seva ajuda per així accelerar el procés.
- R2. El fet d'organitzar dues competicions al mateix temps genera mancances en el material de manera que pot impossibilitar la realització d'una tasca en molts casos. Doncs si un dels equips surt de la universitat per competir necessiten material per emportar-se. Per tant és important que a principis de temporada es faciliti un mínim de material per ambdós equips de tal manera que durant les fases de competició no faltin eines per poder prosseguir. En tot cas sempre es pot intentar disposar de préstecs de material d'altres associacions.

2. Introducció

ROS és un sistema que ofereix moltes prestacions, i a més és *open-source*. És a dir tothom pot accedir a ROS, tothom és lliure de fer i desfer amb el sistema. És per això que ara per ara ROS es constitueix d'enginyeria modular, és a dir que en el cas de necessitar una funcionalitat concreta molt probablement ja estigui dissenyada i realitzada. En el cas d'aquest projecte es parteix d'un *package* el qual conté les funcions que es volen realitzar, a partir d'aquest es tracte d'estudiar-lo extraient-ne el grau de funcionalitat. Així a partir d'aquest es proposa una nova configuració que afavoreix a l'objectiu. Aquest recurs base s'extreu de Github un recurs web on s'hi troben multitud de recursos *open-source*.

2.1. Objectius del treball

L'objectiu d'aquest treball tracte en arribar a una solució que sigui òptima en quant a la competició es refereix, en altres paraules el que es vol és poder aportar al robot modelat en 3D unes simulacions que siguin afins al seu comportament i que representin en el màxim del possible un cas real.

Encara més es vol aconseguir trobar l'algoritme de *path planning* més adequat al robot per tal de que aquest es pugui desplaçar de manera autònoma per l'espai evitant els obstacles sense dificultats.

2.2. Abast del treball

Aquest projecte es centra en ajustar al màxim el *package* mencionat, per fer-ho es disposa de ROS, en concret d'una de les seves distribucions, Kinetic, que treballa amb Ubuntu 16 de Linux. En aquest sentit el treball està limitat per la distribució escollida, les seves restriccions i opcions per defecte. No es disposa de cap element afegit a ella a part del *package* del qual es parteix. A més a més l'abast del treball queda limitat pel reglament de la competició en certs aspectes en quant a disseny del robot i programació es refereix.

3. Evolució del Projecte

PUCRA com a associació d'estudiants dedicada a la robòtica es centra en l'aprenentatge i la cooperació entre els membres per aconseguir un objectiu comú. Partint d'aquest concepte, el seu principal objectiu es involucrar-se en competicions de robòtica reconegudes així com VEX, *Robot Wars* i Eurobot entre d'altres per tal de marcar-se una fita i intentar aconseguir-la. Entre les anteriors competicions esmentades, amb Eurobot es pretenia portar a terme un projecte ambiciós.

Per tenir una mica de context, aquesta competició és un conjunt de partides classificatòries en les quals dos equips deixen anar els seus robots autònoms per a que aquests, en un temps limitat, aconsegueixin la màxima puntuació. Tot i les múltiples opcions de poder construir un robot que mínimament es mogui per un espai tancat conegut i faci diverses accions amb certs objectes, des de PUCRA es va voler optar per l'opció de ROS com a base de software.

Amb el coneixement de que el projecte comportava certs riscos des de l'inici, es va decidir prosseguir amb aquesta idea.

3.1. Idea principal i evolució d'aquesta

Inicialment, l'objectiu del projecte era construir un robot que incorporés una càmera de profunditat (*Depth camera*), un LIDAR (*Light Detection And Ranging*), un ordinador per integrar-hi ROS i així executar el codi font i una *Raspberry* per controlar tot el hardware. Partint d'aquesta base, la tasca que a hom li pertocava fer era portar a terme simulacions amb Gazebo i a l'hora simulacions a temps real ensenyant al robot com moure's pel camp – és a dir, connectant el robot virtual de Gazebo amb el real de tal manera que actuessin igual i a la vegada –, en altres paraules *Machine Learning*.

Així doncs es va emprendre una recerca de informació per tal d'assentar al màxim possible el coneixement requerit amb el conjunt d'integrants del projecte. En cert punt d'aquesta investigació es va decidir abandonar la idea doncs ja s'havia sobrepassat el màxim de temps òptim a dedicar-hi i els avenços havien sigut escassos.

Tot i continuar treballant amb el simulador Gazebo es va prescindir del *Machine Learning* apostant per un robot programat per localitzar-se i fer diverses tasques. A un mes vista de les rondes classificatòries es va trobar que cert material no complia amb certs requisits per poder fer-lo servir. D'aquí que tot el projecte, no només el departament de programació, hagués de fer molts canvis.

Per tant, com a última opció tot lo referent a ROS va desaparèixer i es va passar a Arduino. Per diversos motius, es va finalitzar un projecte que es va començar coneixent que hi havia riscos. Tot i així en aquest moment hom va trobar com continuar endavant.

3.2. Exposició i raonament de la solució final

D'alguna manera s'havia de continuar aquest projecte, però com és evident després de que aquest finalitzés l'associació PUCRA es va centrar en un altre objectiu. Per tant, es va decidir canviar mínimament el plantejament del treball. Ara des de fora de l'associació, aquesta nova solució es va tirar endavant.

Partint de la mateixa base, el càlcul de trajectòries, es va decidir continuar amb les simulacions portant a terme un protocol de proves per arribar concloent amb un *package* a partir del qual es pogués fer *gmapping* amb un robot i un món personalitzats, afí al que es va portar a terme en l'associació. A més de poder, a partir de lo anterior, fer que el robot es dirigís de forma autònoma pel món creat gràcies al AMCL i al *path planning* o càlcul de trajectòries. Doncs el que es planteja per al treball és una introducció a ROS i a la competició d'Eurobot, per després descriure la funcionalitat i funcionament del *package* en qüestió i així exposar el protocol de proves que s'ha dut a terme per arribar al resultat final.

4. ROS

Per arribar a comprendre el que s'exposa en el projecte i les diferents paraules clau, abans es creu convenient introduir al lector en ROS, de manera breu i concisa doncs aquest no és l'únic tema principal del treball.

4.1. Què és i com funciona?

ROS o *Robot Operating System* podria entendre's com a sistema operatiu tal com el seu nom deixa intuir, ara bé no és sinó un *framework* especialment creat per robots que permet gestionar dades de hardware, processos, comunicacions entre blocs, etc.

ROS no funciona per sí sol, és dependent d'un sistema operatiu com a tal, doncs com indiquen els seus propis creadors:

"Ros no és un sistema operatiu en el tradicional sentit de procés de gestió i programació; En canvi, ROS proveeix una capa de comunicacions estructurada la qual queda per sobre del sistema operatiu que actua com a host en un clúster de càlcul heterogeni[1]."(Quigley *et al.*, 2010)

Ara ja s'ha respost què és ROS. El com funciona comença per explicar que ROS proporciona, a part de la seva estructura *open-source*, uns programes de simulació 3D per poder testejar el que bonament es vulgui abans de fer-ho en la realitat, aquests programes són Gazebo i Rviz. Es mostra un exemple d'aquests simuladors en les següents figures.

[1] Clúster de càlcul heterogeni, referint-se a un conjunt de processadors que treballen a l'uníson com si fossin una única unitat de computació, cada un dels processadors pot ser del mateix tipus que el principal o simplement ser un coprocessador realitzant petites tasques com aritmètica de coma flotant per exemple.

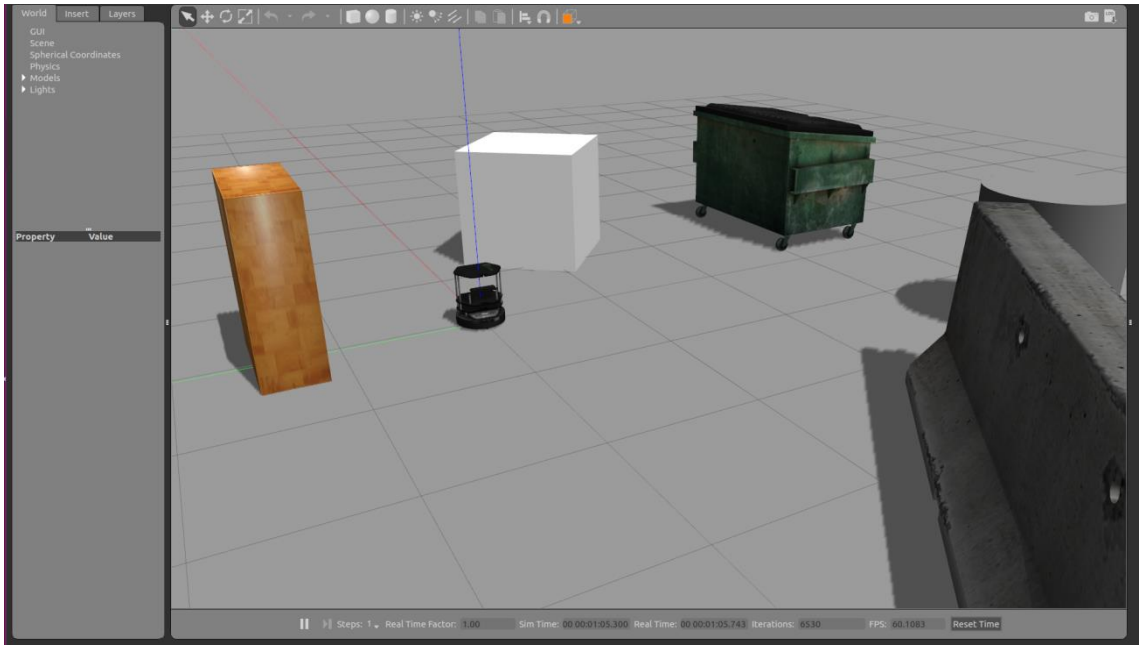

Figura 4.1 Aquest és el robot anomenat *Turtlebot* del que la companyia de ROS n'és el creador. *Turtlebot* es troba en aquesta imatge en el seu món per defecte. La interfície capaç de mostrar aquest escenari en 3D és Gazebo. (Font pròpia)

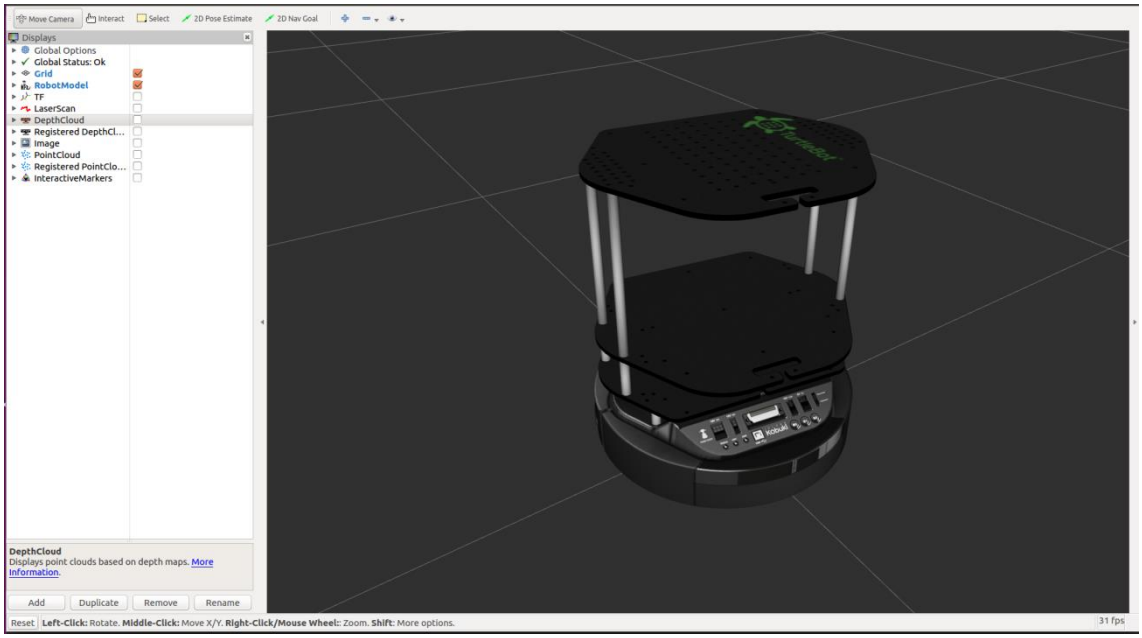

Figura 4.2 Altre cop el robot *Turtlebot* aquesta vegada mostrant-se des de la interfície de Rviz. (Font pròpia)

Aquí arriba el punt de les preguntes, com ha arribat el robot dins del programa? Com s'ha creat el robot? Com es mou el robot?

De moment es pot contestar l'última pregunta, i és que per moure el robot, ROS disposa d'aplicacions de tot tipus. Per fer funcionar una que pugui moure el robot cal executar un tipus

d'arxiu anomenat *launch file*, i al seu torn per fer córrer aquest arxiu cal utilitzar la finestra de comandes de Linux i escriure-hi:

```
roslaunch <package_name> <launch_file_name>.launch
```

Roslaunch és una de les comandes més utilitzades i serveix per executar l'arxiu *.launch* que s'escrigui al final de la comanda. Aquest tipus d'arxiu per a que funcioni ha d'estar dins d'un *package*, aquests són contenidors d'arxius que entre tots formen una aplicació que pot tenir diversos *.launch*, és a dir que pot tenir diverses funcions.

Alguns dels arxius que hi contenen són metadades que es creen juntament amb el *package*, la resta són arxius que hom pot ficar-hi. Pel que fa a un *.launch*, s'hi pot trobar més d'una referència als nodes (Figura 4.3).

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>

  <param name="object_description" command="$(find xacro)/xacro --inorder $(find rosbot_description)/
urdf/EnemyRobot1.xacro"/>

  <node name="object_spawn" pkg="gazebo_ros" type="spawn_model" output="screen" args="-urdf -param
object_description -x 0.2 -y 1 -model enemy" />

</launch>
```

Figura 4.3 Exemple d'un arxiu executable *launch*. (Font pròpia)

En essència aquests nodes són receptors i/o missatgers de tòpics que contenen informació; els *launch files* són arxius executables que serveixen per posar en marxa els nodes. I en tant als tòpics, aquesta informació que contenen esta empaquetada en missatges els quals són un conjunt de diferents tipus de dades i/o altres missatges. Un tipus de missatge és únic però un tòpic pot ser qualsevol tipus de missatge. En la figura següent es mostren dos tòpics molt utilitzats amb els seus tipus de missatges: per una banda */odom* és un tòpic bastant complert representat pel missatge *nav_msgs/Odometry* que conté els vectors de velocitat tant linear com angular, així com també la posició amb covariància del robot; I per altre banda el tòpic /tf és el tòpic que relaciona tots els punts

d'origen dels elements en l'espai i està representat per *tf2_msgs/TFMessage* que conté la relació entre orígens en format de translació i quaternió[1].

```
magloe@TFG:~$ rostopic info /odom
Type: nav_msgs/Odometry

Publishers:
 * /gazebo (http://TFG:33101/)

Subscribers:
 * /move_base (http://TFG:34003/)


magloe@TFG:~$ rosmsg info nav_msgs/Odometry
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

```
magloe@TFG:~$ rostopic info /tf
Type: tf2_msgs/TFMessage

Publishers:
 * /robot_state_publisher (http://TFG:38229/)
 * /amcl (http://TFG:36171/)
 * /gazebo (http://TFG:33101/)

Subscribers:
 * /move_base (http://TFG:34003/)
 * /amcl (http://TFG:36171/)
 * /rviz (http://TFG:38801/)


magloe@TFG:~$ rosmsg info tf2_msgs/TFMessage
geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  string child_frame_id
  geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion rotation
      float64 x
      float64 y
      float64 z
      float64 w
```

Figura 4.4 Exemple de dos tòpics diferents amb els seus tipus de missatges. A l'esquerra el tòpic */odom* de tipus *nav_msgs/Odometry*. A la dreta el tòpic */tf* de tipus *tf2_msgs/TFMessage*. (Font pròpia)

Així en essència, ROS és una estructura de nodes que parlen entre ells, és a dir com una estructura *master-slave*. I és que el màster en aquest cas és *roscore*:

"*roscore* és una col·lecció de nodes i programes que són un pre-requisit d'un sistema ROS." (Open Source Robotic Fundation, 2011)

[1] En l'àmbit de la robòtica un quaternió és un format que representa la rotació i orientació amb un vector de 4 dimensions (x, y, z, w). Aquest format és fet servir per evitar el *gimbal lock*, un problema present en els angles d'Euler que dificulta la lectura dels angles en l'eix Y a l'hora d'apropar-se als ±90 graus.

Per tant per fer servir el sistema de ROS abans és necessari inicialitzar-lo, *roscore* s'encarrega de fer-ho i a més és el node màster que controla el sistema en sí.

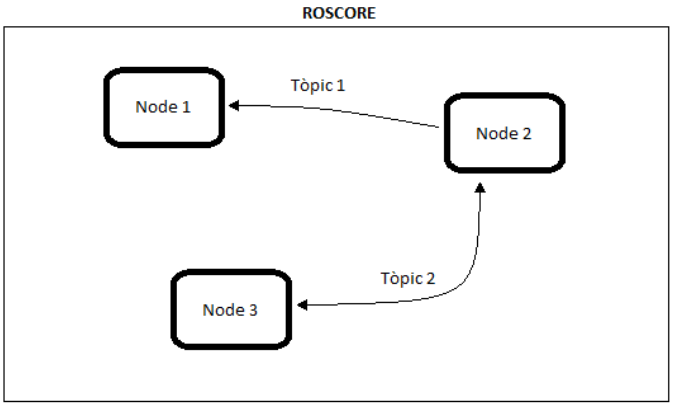

Figura 4.5 Representació de *roscore*. (Font pròpia)

4.1.1. Publishers i Subscribers

Tot i que en definitiva els nodes es poden veure com receptors i missatgers d'informació en realitat són programes escrits en Python o C++, els *Publishers* i *Subscribers* doncs són objectes del sistema de ROS que s'encarreguen d'enviar i rebre la informació. En tant que un node pot ser senzillament programat amb un d'aquests elements, es pot entendre que hi ha dos tipus de nodes, tot i no ser estrictament correcte doncs pot contenir més d'un *Publisher* i un *Subscriber*.

El primer tipus mencionat és el node capaç de publicar informació al sistema i l'altre tipus és qui la rep, val a dir que els nodes no s'envien informació directament entre sí, *roscore* és l'encarregat de controlar aquests missatges. Mentre un *Publisher* publica informació a la xarxa, *roscore* la gestiona i si hi ha cap *subscriber* que la reclami és el mateix *roscore* qui l'entrega. D'aquesta manera l'usuari pot tenir coneixement total de la informació i processos actius doncs tot passa pel màster.

En la Figura 4.6 es mostra com es defineixen el node i els elements publicador i subscriptor, ambdós elements comparteixen el tòpic /cmd_vel i és que un n'envia informació sobre aquest i l'altre en rep, però és només un tòpic per cada element d'aquests declarat. Cada un d'ells té un comportament diferent, un de tipus publicador és com un port sèrie, el qual primer es defineix (indicant el tòpic i el tipus de dada) i després simplement es publica la informació. Pel que fa a un node de tipus subscriptor, aquest es defineix com una interrupció del programa que salta quan es rep informació sobre un tòpic en concret i fa executar la funció d'interrupció corresponent.

```python
#!/usr/bin/env python

import rospy
from geometry_msgs.msg import Twist

class SpeedTest(object):
    def __init__(self):
        self.velocity_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
        self.velocity_subscriber = rospy.Subscriber('/cmd_vel', Twist, callback)
        self.rate = rospy.Rate(1)

    def Start(self):
        while not rospy.is_shutdown():
            Twist vel
            for i in range(1, 5):
                xLinearVel = i/10
                zAngularVel = i*7/10
                vel.linear.x = xLinearVel
                vel.angular.z = zAngularVel
                self.velocity_publisher.Publish(vel)
                self.rate.sleep()

    def callback(self, msg):
        rospy.loginfo(msg.data)

if __name__=='__main__':
    rospy.init_node('spiral', anonymous = True, log_level = rospy.INFO, disable_signals = False)
    classobject = SpeedTest()
    try:
        classobject.Start()
    except rospy.ROSInterruptException:
        pass
```

Figura 4.6 Exemple d'un programa escrit en python on s'hi mostra un *Publisher* i un *Subscriber*. (Font pròpia)

4.1.2. Actions i Services

La gran majoria de nodes funcionen a partir de publicadors i subscriptors de tòpics, ara bé ROS incorpora un altre parell de recursos per gestionar els programes i les dades que s'envien. Es tracte de les *Actions* i els *Services*. De forma resumida un *service* ve a ser com una interrupció del curs del programa, és tracte d'un altre programa que quan està en execució res més està en marxa fins que el *service* hagi acabat. Aquest procés és útil si per exemple, en el context d'aquest projecte: el robot troba amb la càmera una peça, para tots els processos i executa un *service* per reconèixer la peça; fins que el robot no hagi decidit si li interessa, no farà cap altre cosa doncs necessita estar ben quiet per que la càmera funcioni amb la seva màxima resolució.

Per altre banda una *action* és asíncrona, a diferència d'un *service* que és síncron; ja que es tracte d'un programa que s'executa mentre altres també ho fan. Com per exemple el posicionament del robot, doncs constantment s'ha de poder saber on és el robot i respecte a què sense haver de parar altres processos.

4.2. Què són Gazebo i Rviz?

Rviz per ROS *visualization* és l'eina per defecte de ROS per poder visualitzar el robot i fer simulacions. Per altre banda Gazebo neix apart i és incorporat més tard a ROS sent el simulador 3D de preferència. Ambdós són eines de visualització i simulació, tot i que tenen funcions diferents.

Per una part, Rviz ha perdut part de la seva finalitat de visualització del robot, tot i així és altament funcional ja que connecta directament amb ROS, concretament amb tots els nodes i tòpics que estan actius. En tot lo referent al càlcul de trajectòries que aquest projecte abraça, Rviz pren molt de protagonisme ja que és l'encarregat de mostrar les trajectòries calculades, el punt de vista de la càmera, el camp tal i com s'ha escanejat, els punts que el LIDAR veu, etc. Aquí és mencionen algunes de les característiques que aquest simulador té, en canvi la principal característica són els *markers*: són objectes primitius que permeten visualitzar la informació que hom vol sense que aquesta hagi de ser interpretada per Rviz, tenen la capacitat de ser interactius fins al punt de poder arribar a crear un joc retro amb aquests *markers*.

Per altre banda Gazebo ha pres el protagonisme en quant a simulació i visualització, darrera d'aquesta eina gràfica hi treballa un potent motor de gràfiques i físiques que és afí al comportament de l'entorn fora de la simulació, a més és capaç de generar dades provinents de sensors afegint el soroll que hom cregui convenient per fer la simulació més realista.

Tant per Rviz com per Gazebo l'addició de models 3D es fa de la mateixa manera, si més no semblant. Hi ha desenes de tipus d'arxius que defineixen a un model 3D, però normalment són específics per una eina gràfica en concret, per exemple SolidWorks treballa de base amb arxius del tipus SLDPRT. Pel que fa a Gazebo i Rviz ambdós treballen amb un tipus d'arxiu base tant si el model és exportat com si no, ha d'acabar sent definit en format SDF. Aquest tipus d'arxiu s'escriu en llenguatge XML, en el que s'hi descriuen: les peces que composen el model i les seves físiques així com els moments d'inèrcia, els punts d'origen i referència i els coeficients de fricció entre d'altres; i les unions entre peces que han de ser d'un tipus d'entre les que disposen tant Gazebo com Rviz. Tot i així, altres arxius com URDF i XACRO també en format XML són acceptats, un darrera l'altre són la simplificació de l'anterior.

URDF és un únic arxiu com SDF que a diferència d'aquest últim no s'hi defineixen tantes físiques, cosa que així se'n redueix la llargada de l'arxiu. Pel que fa a XACRO, aquest normalment són més d'un fitxer i té la mateixa estructura que URDF però incorpora una llibreria matemàtica que permet realitzar operacions per simplificar-ne la definició.

L'estructura d'un URDF es basa en les peces que incorpora el model, anomenats *links*, i les unions anomenades *joints*. El model que es mostra en la següent figura representa dos cubs, un la meitat de

l'altre, units per una *joint* en l'eix z de tipus *continuous*: això és que per un dels tres eixos prèviament seleccionat tant una peça com l'altre es poden moure al voltant d'aquest eix. La Figura 4.8 representa la definició del model en un arxiu URDF.

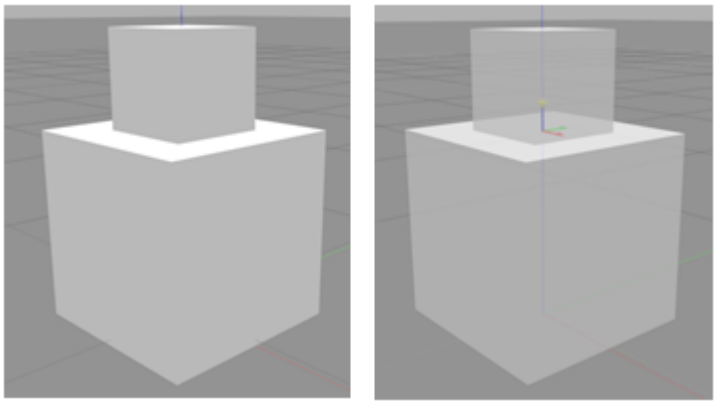

Figura 4.7 A l'esquerra el model 3D d'exemple. A la dreta el mateix model en visió transparent mostrant la posició de la *joint*. En groc l'eix pel qual les dues peces estan unides. (Font pròpia)

```
<?xml version="1.0"?>
<robot name="box">

  <link name="base_link">
   <visual>
     <origin rpy="0 0 0" xyz="0 0 0"/>
     <geometry>
       <box size="0.5 0.5 0.5"/>
     </geometry>
   </visual>
   <collision>
     <origin rpy="0 0 0" xyz="0 0 0"/>
     <geometry>
       <box size="0.5 0.5 0.5"/>
     </geometry>
   </collision>
   <inertial>
     <mass value="10"/>
     <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
   </inertial>
  </link>

  <joint name="base_head" type="continuous">
    <axis rpy="0 0 0" xyz="0 0 1"/>
    <parent link="base_link"/>
    <child link="head"/>
    <origin rpy="0 0 0" xyz="0 0 0.25"/>
  </joint>

  <link name="head">
   <visual>
     <origin rpy="0 0 0" xyz="0 0 0.125"/>
     <geometry>
       <box size="0.25 0.25 0.25"/>
     </geometry>
   </visual>
   <collision>
     <origin rpy="0 0 0" xyz="0 0 0.125"/>
     <geometry>
       <box size="0.25 0.25 0.25"/>
     </geometry>
   </collision>
   <inertial>
     <mass value="5"/>
     <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
   </inertial>
  </link>

  <plugin name="plannar_mover_plugin" filename="libplannar_mover_plugin.so"/>
</robot>
```

Figura 4.8 URDF senzill d'exemple. (Font pròpia)

En la figura anterior també s'hi observa l'apartat *plugin* que en resum és un afegit, una característica o funció que se li afegeix a la definició del robot. Un *plugin* s'associa amb una peça o *link* del robot, per exemple amb les rodes se li afegeixen moviment i als sensors se'ls hi afegeix la característica de poder entregar dades i llegir-ne. En aquest cas es tracte d'un *plugin* que permet fer moure la base del model, juntament amb tots els seus *links* associats, de forma holonòmica sense necessitat de rodes.

Altres aspectes a esmentar són les físiques definides en aquest arxiu. Com s'ha comentat anteriorment URDF es basa en SDF però reduint-ne certes característiques. Realment són el mateix arxiu però URDF no obliga a definir tants aspectes com SDF, el que s'observa en la figura anterior no és el mínim que se'n podria escriure d'un model 3D, bastaria amb la geometria i la col·lisió. A l'hora d'executar el *launch* que fa aparèixer el model, Gazebo directament s'encarrega d'omplir els buits dels d'aspectes no definits amb zeros. Per exemple, aquest model té una unió entre les seves dues peces però en la Figura 4.8 no s'indica el fregament entre blocs; per tant, si s'aplica una força per fer girar el bloc més petit sobre el gran, aquest primer no deixaria de girar.

5. Eurobot, la competició

Eurobot, una competició de robòtica que des de 1998 promou la participació i l'aprenentatge en aquest àmbit. Des de PUCRA es comparteixen aquests objectius és per això que l'associació va decidir involucrar-s'hi.

Aquest projecte s'ha dut a terme en torn a aquesta competició, doncs en aquests apartats que segueixen s'introdueix al lector tant en el funcionament bàsic d'Eurobot com en la temàtica i reglament de l'any en que s'ha realitzat tal treball.

5.1. Funcionament bàsic de la competició

Eurobot és una competició internacional, hi participen equips de països provinents des de Espanya fins a Rússia o Tunísia i Algèria. Cada equip participant ha de primer competir en el seu respectiu país i els tres primers classificats de cada país arriben a la final que es disputa a França la qual sempre n'ha sigut l'amfitriona d'Eurobot.

El campionat no és una lluita de robots com és lo habitual pensar, en canvi tracte de complir certs objectius que canvien amb l'edició i la temàtica. Objectius tals com: recollir pilotes d'un punt determinat del camp de joc i deixar-les en algun altre en concret; apilar cubs en una zona determinada i segons els seus colors, o recopilar discos de pes diferent per tal de guanyar una lluita en una balança. A més, una de les característiques principals d'Eurobot és que els robots han de ser completament autònoms.

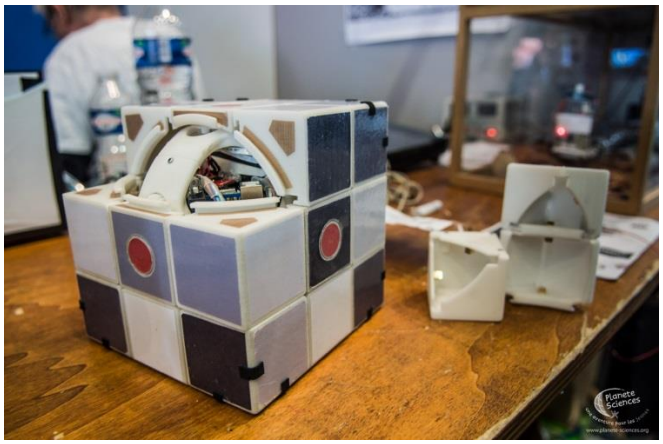

Figura 5.1 Exemple d'un robot creat per un dels equips francès participant en la final internacional (Font: (Eurobot, 2019a))

A finals de cada any es fa saber el tema i el reglament del joc de l'any vinent, d'aquesta manera els equips participants poden començar a elaborar les seves idees. A principis d'any es donen a conèixer les ubicacions, dates i participants de cada país per dur a terme les rondes classificatòries, normalment des de que es fa saber aquesta informació hi ha un temps d'uns 2-3 mesos fins a les rondes de classificació per països, i un mes més tard es disputa la final.

El caràcter d'aquestes rondes és idèntic a les que es realitzen en la final per tant es passarà a explicar-ne el seu contingut. Abans de començar qualsevol partida, en la final a França i en les classificatòries de cada país, els equips són cridats pels jutges o àrbitres per dur a terme les homologacions dels robots. En aquesta secció de la competició, els robots són avaluats per diferents aspectes, de tal manera que si no es compleixen tots els requisits tal equip no pot participar. En el següent apartat es donen a conèixer dades més específiques sobre els requisits dels robots en l'edició 2019.

A partir d'aquí es juguen partides classificatòries fins que només queden dos equips que es disputen el títol de primer finalista de l'edició. En cada partida ambdós equips deixen anar els seus robots en el compte enrere establert per l'àrbitre de taula. Un cop acabat el temps, si és que no hi ha hagut cap eliminació d'algun dels equips, els àrbitres compten els punts totals que ha fet cada participant en la partida i es proclama un guanyador.

En total, tenint en compte les rondes prèvies a la final segons el nombre de participants en aquestes, Eurobot pot durar entre 3 i 6 dies.

Figura 5.2 Robot en plena partida de la final internacional (2013), llençant una pilota de ping-pong a una cistella (Font: (Eurobot, 2019a))

5.2. Regles i restriccions bàsiques

Figura 5.3 Camp de joc de l'edició 2019 d'Eurobot, *Atom Factory* (Font: (Eurobot, 2019b))

Atom Factory és el nom que pren la temàtica de l'edició 2019 d'Eurobot. En la Figura 5.3 s'hi observa el camp de joc, es tracte d'un camp de 3 x 2m on s'hi distingeixen discs d'hoquei amb diferents colors i les diferents zones que prenen noms acords amb la temàtica, com per exemple l'accelerador de partícules o la taula periòdica. El camp es divideix en dues parts una per cada equip, els colors grocs i lila són distintius de cada un d'ells. De forma resumida el que es tracte d'aconseguir en la partida és agrupar tants discos com es sigui possible i ubicar-los tant en la taula periòdica (separant els discos per colors) o en la bàscula (separant els discos per pes). Per tant en *Atom Factory* només existeix un tipus d'objecte, el disc d'hoquei; ara bé, aquests poden ser de diferent pes i color:

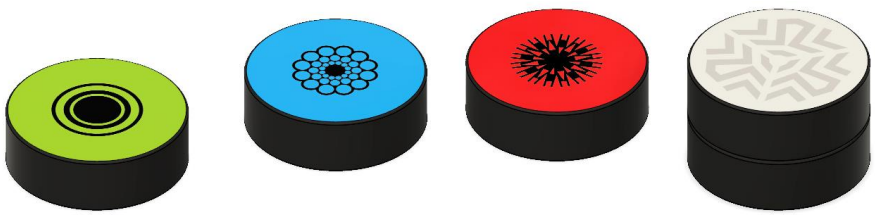

Figura 5.4 Elements del joc, àtoms representats per discs d'hoquei (Font: (Eurobot, 2019b))

- “Greenium” (disc verd), amb un total de 12 discs al camp tenen un pes de 120 g.
- “Bluenium” (disc blau), amb un total de 8 discs al camp tenen un pes de 170 g.
- “Redium” (disc vermell), amb un total de 16 discs al camp tenen un pes de 60 g.
- “Goldenium” (disc blanc), amb un total de 2 discs al camp tenen un pes de 340 g.

Ara bé, segons on es dipositin els discs i de quina manera la puntuació varia, però els punts també varien en funció de les accions que es poden realitzar al camp en certes zones. Aquestes zones i les demès són les que es llisten a continuació:

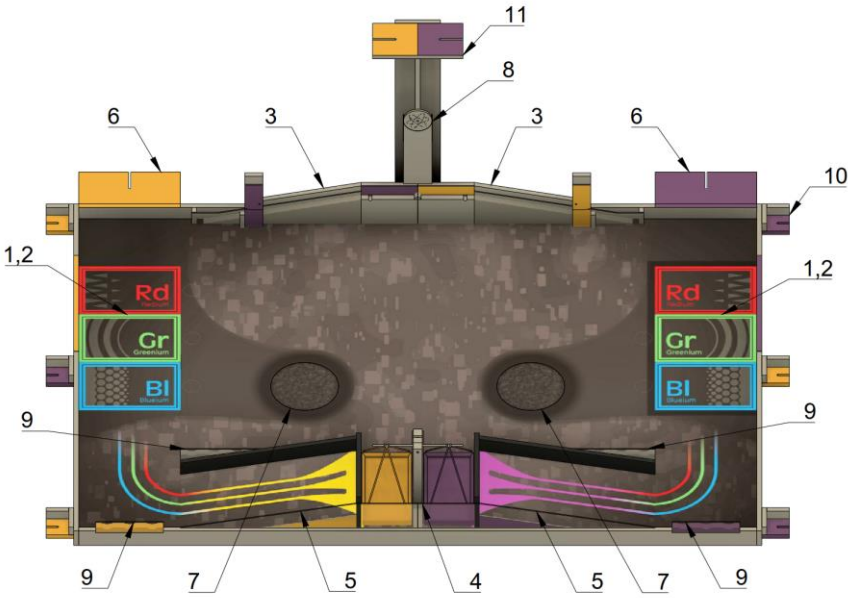

Figura 5.5 El camp de joc amb les diferents zones marcades (Font: (Eurobot, 2019b))

1. Taula periòdica. Cada casella de colors correspon amb un tipus d’àtom, n’hi ha 3 per equip i són exclusives.
2. Zona d’inici. La funció d’aquestes caselles també és la de punt de partida dels robots, i aquests no poden revessar-les.
3. Accelerador de partícules. Fent lliscar un disc per la rampa s’acciona un mecanisme que fa caure el “Goldenium”, permetent llavors recollir-lo.
4. Bàscula. Aquí es poden dipositar els discos i competir per veure quin equip aconsegueix un major pes, tenint en compte que només s’hi poden col·locar fins a 6 discs.
5. Rampa. És la rampa d’accés a la bàscula tot i que no és obligatori accedir-hi per aquesta via.
6. Zona de l’experiment. Com cada any existeix un repte a més a més dels robots a construir, això és l’experiment, que més endavant s’explica.

7. Zona de caos. N'hi ha dos al camp i els dos equips hi poden accedir ja que no porta cap color distintiu d'equip, en aquesta zona hi ha 4 discs aleatòriament col·locats tot i que sempre són: dos "Redium", un "Greenium" i un "Bluenium".
8. Àtom d'oxigen. Forma part de l'experiment
9. Distribuïdor d'àtoms. N'hi ha 4, dos de comuns (al centre del camp) i la resta un per a cada equip (a les cantonades davant de les rampes). Són suports horitzontals on els discs s'hi ubiquen de forma vertical.
10. Suports fixes pels *beacons*. N'hi ha 3 per a cada equip (dos en cada extrem de la meitat del camp propi de l'equip i per fora d'aquest, i l'altre al camp contrari entre mig dels dos anteriors), cada un d'ells ha de construir els seus propis *beacons* en cas de voler utilitzar-los, més endavant se n'explicarà la utilitat.
11. Torre central de seguiment. Serveix per ubicar-hi els dispositius necessaris per fer la funció de *tracking* o seguiment sempre i quan aquests dispositius no revessin l'àrea plana de la torre. De la mateixa manera que els *beacons*, col·locar aquests dispositius és opcional.

Tots els anys el camp d'Eurobot no varia en certs aspectes, un d'ells la grandària del camp que sempre és de 3 per 2 metres, una altre és la torre central de seguiment o *tracking*, un altre són les bases per col·locar els *beacons* i l'última és l'àrea de l'experiment. En aquesta base que s'indica en la Figura 5.5 (element 6), s'hi munta l'experiment que cada equip ha construït. Aquest no ha de sobrepassar les mesures de la base i la seva temàtica canvia amb l'edició.

Per aquesta competició en concret es demanava fer un dispositiu que es pogués activar (tant de manera remota des del robot com "manual", és a dir activat pel robot amb alguna part dels seu cos), i que a l'hora de fer-ho un electró recorreria el camí des de l'experiment fins a l'àtom d'oxigen quedant-s'hi al seu costat. L'experiment i l'àtom han d'estar connectats per una corda, a partir d'aquí i alguna altre restricció es podia crear el que fos.

5.2.1. Puntuació

Doncs segons com es classifiquin els discs i segons el lloc es donen uns punts o uns altres, comptant que també hi ha punts per l'experiment i altres bonificacions.

Punts de la taula periòdica:

- 1 per cada disc col·locat en qualsevol de les caselles.
- 5 (5+1) addicionals per cada disc col·locat en la casella correcta.
- 6 pel "Goldenium".

Punts de la bàscula:

- 4 per cada "Redium".
- 8 per cada "Greenium".
- 12 per cada "Bluenium".
- 24 pel "Goldenium".

Punts de l'accelerador de partícules:

- 10 per cada àtom present en l'accelerador.
- 10 (10+10x) addicionals per activar el mecanisme.
- 20 (20+10+10x) addicionals per extreure el "Goldenium".

Punts per l'experiment:

- 5 per col·locar-lo en l'àrea de l'experiment abans de començar la partida.
- 15 addicionals (15+5) per activar-lo durant la partida.
- 20 addicionals (20+15+5) per que l'electró hagi arribat a l'oxigen abans d'acabar la partida.

Cada equip té l'opció d'indicar una estimació de la puntuació que farà abans de començar la partida, a partir d'aquesta i la puntuació real és calcula una puntuació de bonificació de la següent manera:

$$Bonus = 0{,}3 \cdot Puntuació - Delta \qquad \textbf{(Eq. 5.1)}$$

Sent *Delta* la diferència en valor absolut entre la puntuació real i l'estimació. *Bonus* és un número enter i no menor a zero.

Punts per bonificacions i altres:

- El *Bonus* s'afegeix a la puntuació al final de cada partida
- -40 per cada penalització
- 30 de bonificació per guanyar la balança (només a partir dels setzens de final)
- 10 per cada equip que no estigui desqualificat
- Un equip desqualificat tindrà una puntuació de 0 punts

5.2.2. Reglament

Hi ha certes decisions que han afectat al resultat final d'aquest treball i és degut a la normativa que imposa Eurobot en la seva competició. En aquest apartat s'hi descriuen les normes considerades més importants i que poden haver significat un canvi.

Normes de caràcter general:

- No tocar cap robot contrari, de forma intencionada o no, en cap altre cas és classifica com una penalització.
- No és permès utilitzar les zones contràries ni els objectes que hi romanen, això inclou: el distribuïdor d'àtoms petit, la taula periòdica i els discs que s'ubiquen just davant, l'experiment, la balança, l'accelerador de partícules i el "Goldenium" pertinent.
- La partida té una durada de 100 segons.

Normes referents a l'homologació, el robot o robots:

- Han d'incorporar un sistema de detecció d'obstacles amb el qual actuar en conseqüència.
- Han d'incorporar un botó d'emergència que aturi tots els actuadors del robot, aquest ha de ser d'unes mesures concretes i si és possible de color vermell.
- Han d'incorporar un suport pels *beacons* dels enemics.

Normes referents a la dimensions dels robots:

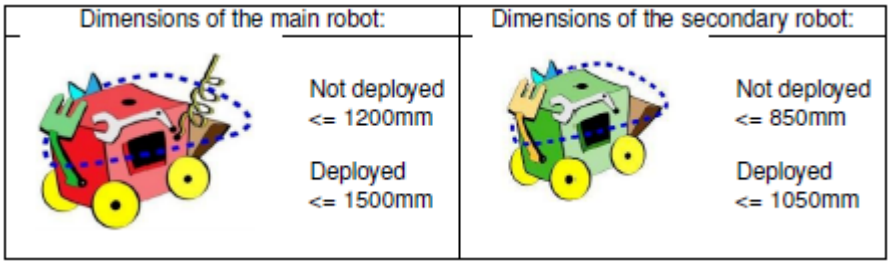

Figura 5.6 Taula que indica les dimensions del robot en mil·límetres (Font: (Eurobot, 2019b))

- Els robots com a màxim han de tenir una alçada de 350 mm sense comptar el suport pels *beacons* dels contrincants.
- El suport pels *beacons* dels contrincants pot començar en qualsevol altura, però la superfície més alta ha d'estar en els 430 mm mesurant des del terra. A més ha de ser robust i capaç de suportar 300 g.

5.3. Presentació del robot, hardware

En aquest apartat és dona a conèixer el robot que es va arribar a fer, el què incorpora i la seva principal funció. Tot i que no és un aspecte del tot rellevant de cara a la conclusió del projecte aquest robot ha servit de base per construir el disseny 3D, és més a més un disseny que pot servir de base de cara a futures millores del projecte actual. La competició de la final europea demanava un pòster en el que es presentés al robot.

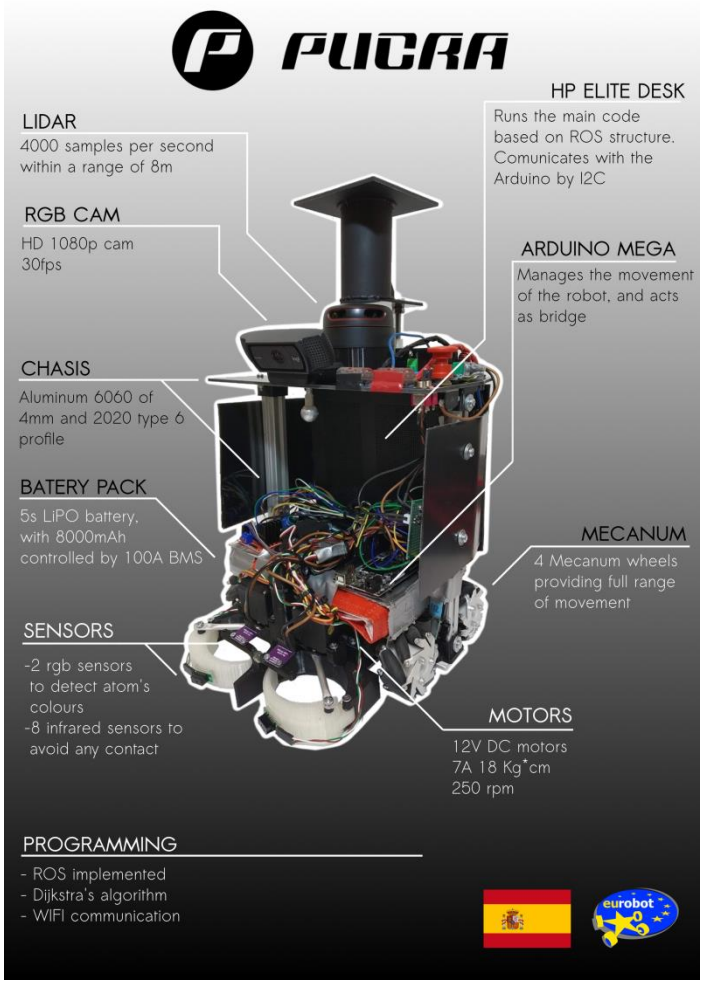

Figura 5.7 Pòster del robot secundari (Font pròpia)

Un dels principals punts a destacar és l'ús de rodes *mecanum*, aquestes permeten una multi varietat de moviments gràcies als petits rodets que envolten la roda. Amb els rodets incorporats en angle al voltant de la superfície de la roda es pot aconseguir dirigir al robot tant diagonalment com lateralment tenint en compte que la resta de moviments convencionals en el pla horitzontal també

és capaç de fer-los, el fet de portar aquest rodets no produeix un lliscament d'aquests mentre la roda gira donat que el propi pes que suporten ho limita.

Les característiques bàsiques del hardware es poden observar fàcilment en la figura anterior, com per exemple el rang d'abast del LIDAR, d'uns 8 metres pel model que es feia servir (RPLIDAR A2M8), a més aquest dispositiu d'escanejat és capaç de captar 4000 mostres per segon podent girar a una velocitat de 600 rpm.

Altres aspectes a destacar són per exemple les bateries LiPO que es van manufacturar a l'associació, estaven fetes amb 5 cel·les de bateries de Liti, entre elles podien entregar fins a aproximadament 18 V; L'estructura mecànica estava feta de planxes d'alumini 6060 de 4 mm d'espessor a més de perfils 2020 d'Alumini amb ranura de 6 mm; les pinces muntades a la part davantera del robot tenen la capacitat d'obrir-se gràcies a uns servos connectats a la placa d'Arduino, la funció d'aquestes pinces era poder atrapar discs de tal manera que els pogués transportar; els sensors de color estan justament col·locats a sobre de les pinces per poder llegir el color un cop s'hagués agafat un disc; els sensors d'infraroig s'utilitzen com a mesura d'emergència en cas de fallar el sistema principal de detecció d'obstacles, que en aquest cas es tractava del conjunt ROS – LIDAR; La càmera que en aquesta Figura 5.7 apareix no és la que en un principi s'havia descrit, en primera instància es tractava d'una *depth càmera* (Intel RealSense D435), la qual després de realitzar-ne diverses proves es va desestimar utilitzar-la degut a la seva baixa resolució, doncs aquesta causava que el reconeixement d'objectes a partir de la visió per computador no funcionés com s'esperava. Per tant, amb varies opcions entre mans es va optar per un càmera RGB de més resolució, amb la qual el reconeixement d'objectes resultava ser més eficaç.

6. Introducció al *Package* i a ROS *Navigation*

L'objectiu final d'aquest treball és donar a conèixer com s'ha portat a terme principalment el calibratge del càlcul de trajectòries i en segon pla la resta de components del *package*, per al seu correcte i desitjat funcionament. Per arribar fins a aquest punt abans s'ha introduït al lector en ROS i la competició d'Eurobot, ara però just abans d'entrar en detall amb les proves fetes referents als *path planning* és necessari conèixer què hi ha darrere, com s'ha elaborat i com funciona.

6.1. Introducció al càlcul de trajectòries

El càlcul de trajectòries o *path planning* és el nom donat a la solució d'un famós problema, el problema del camí més curt. Des de 1958 s'han aportat solucions amb algoritmes com el de Bellman (1958), Dijsktra (1959) o Peter E. Hart et. al. (1968) entre molts d'altres. Els dos últims són els més famosos, l'algoritme de Dijkstra i el de Peter E. Hart més conegut com algoritme A*.

6.1.1. Algoritme Dijkstra

L'algoritme de Dijkstra parteix d'un gràfic amb múltiples nodes, entre els quals existeix un únic camí amb una distància coneguda i dos nodes coneguts: el d'origen i el de destí. Dijkstra suggereix crear dos grups de classificació de nodes, un pels ja visitats i un pels no visitats. Partint del node inicial A es considera que la distancia temptativa a A és 0 i la resta de distancies temptatives tenen valor infinit. A partir d'aquí s'avalua cada node veí de A no visitat i es calcula la distància temptativa. Si ja hi ha un valor de distància temptativa calculat pel node en qüestió, i resulta que el nou càlcul és menor al que ja existia, sempre ha de romandre el menor valor. Partint d'un node, el veí que tingui la distància temptativa més baixa serà el següent visitat, i sempre que es calculi una distància temptativa se'n guardarà amb aquesta el node des d'on s'ha calculat (Djikstra, 1959). En definitiva la distància temptativa es calcula de la següent manera:

$$d_{t_i} = d_t + D(a,i) \qquad \textbf{(Eq. 6.1)}$$

Sent d_{ti} la distància temptativa al node *i*, d_t la distància temptativa del node actual i D(a, i) la distància entre el node actual i el node *i*. Aquest seria doncs l'algoritme bàsic de Dijkstra, en l'actualitat existeixen multitud de versions modificades del mateix algoritme. En les figures següents es mostra un exemple de com funciona l'algoritme de forma gràfica.

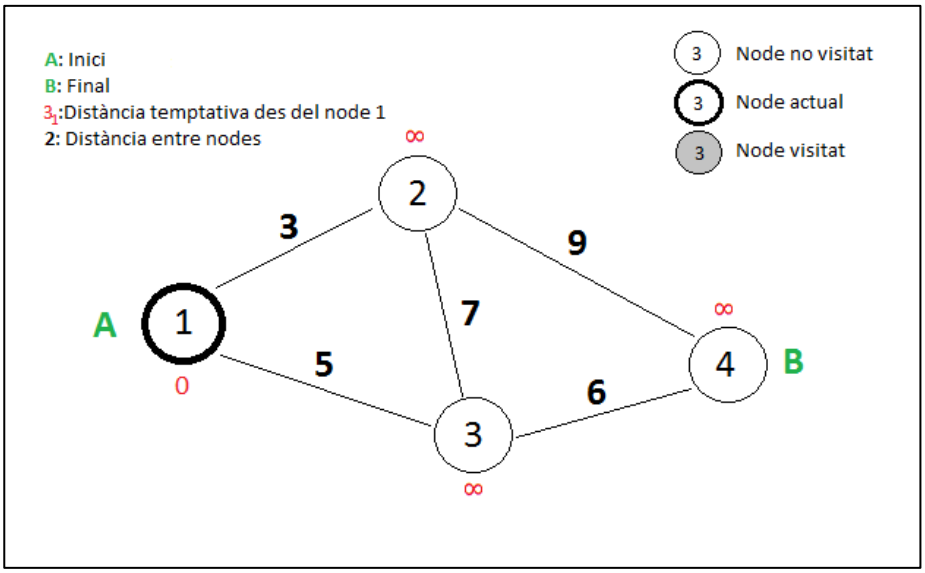

Figura 6.1 Es comença en el node 1 sent aquest l'actual i amb una distancia temptativa de zero, les demès distàncies temptatives tenen valor infinit. (Font pròpia)

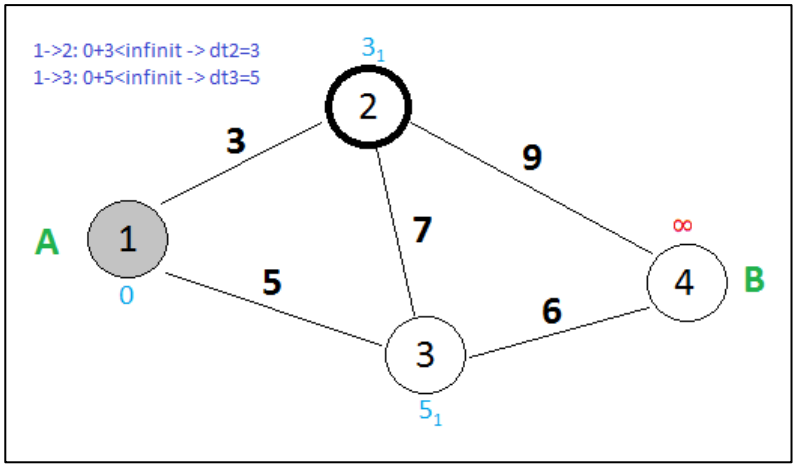

Figura 6.2 Es calculen les distàncies temptatives dels veïns del node 1 indicant d'on es prové, en aquest cas del node 1. Aquest queda com a visitat i l'actual passa a ser el de la distància mínima és a dir el node 2. (Font pròpia)

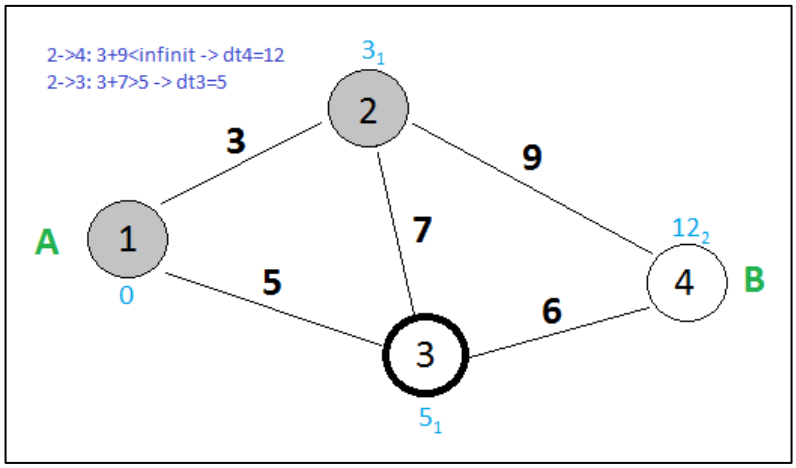

Figura 6.3 Resulta que des del node 2 fins al 3 la distància temptativa és de 10 sent més gran a la que ja s'havia calculat, per tant es conserva la distància temptativa del node 3, és a dir 5 provinent del node 1. El node 3 passa a ser el node actual per tenir la distància mínima mentre que el node 2 queda com a visitat. (Font pròpia)

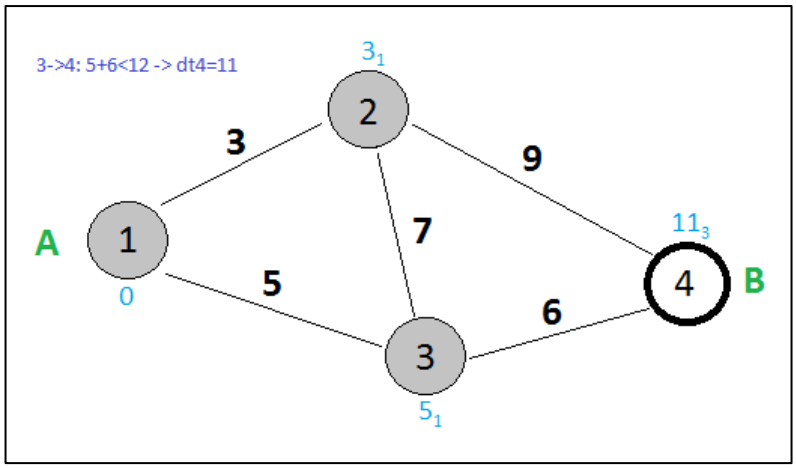

Figura 6.4 Només queda un node no visitat i n`és el de destí B. La distància temptativa des del node 3 és 11, menor a la que s'havia calculat des del node 2, és a dir 12, per tant es canvia. Aquí acaba l'algoritme al no trobar més nodes no visitats, i si es ressegueixen els nodes previs des del final fins a l'inici es troba el camí més curt, en aquest cas és 1-3-4. (Font pròpia)

Aquesta és un demostració en format grafo, ara bé tenint en compte que el càlcul es durà a terme en un mapa, els nodes canvien i passen a tenir una distància entre ells igual. Com en el cas que es mostra en la Figura 6.5: el punt verd representa el node d'inici i el vermell el de destí; la resta de nodes blancs són desconeguts i la zona de color negre és un obstacle. Es desconeix la distància entre nodes però es pot suposar que tinguin un valor de 1 les distàncies tant verticals com horitzontals, per tant en diagonal resulta tenir un valor de $\sqrt{2}$. En un principi l'algoritme començarà per un dels 4 veïns més propers (amunt, avall, a l'esquerra i a la dreta del punt verd), s'expandirà per un d'ells i a mesura que vegi que els costos són iguals o majors als nodes que ha deixat enrere, doncs estudiarà aquests nodes anteriors. Finalment, el resultat serà semblant al representat en la Figura 6.6: amb els

nodes blaus com a estudiats o visitats, els blancs com a desconeguts o no visitats i els nodes lila són el camí més curt que s'ha trobat.

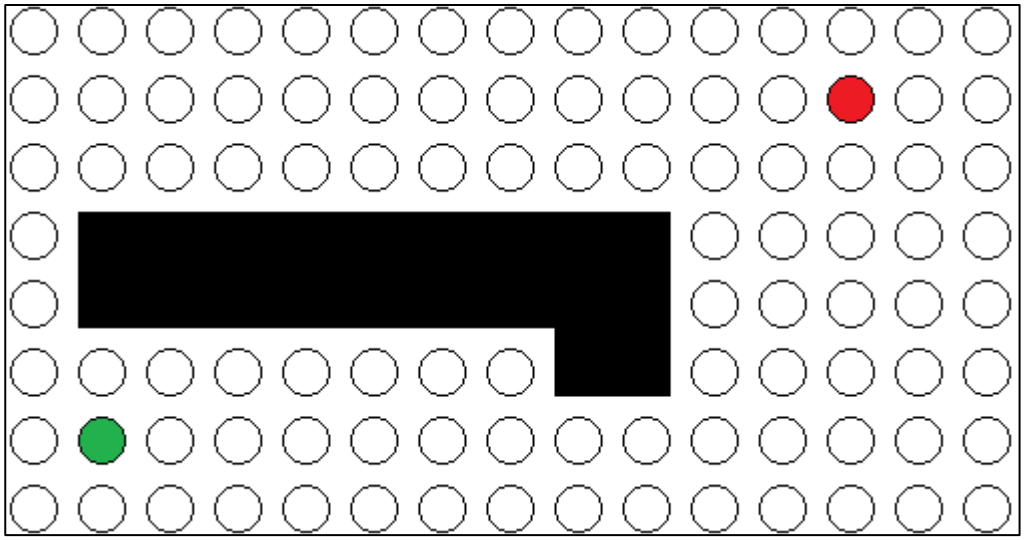

Figura 6.5 Exemple de l'aplicació de Dijkstra en un mapa de punts formant una quadrícula (1). El node verd és l'inici i el vermell el destí, la resta de nodes blancs són desconeguts i el bloc negre és un obstacle. (Font pròpia)

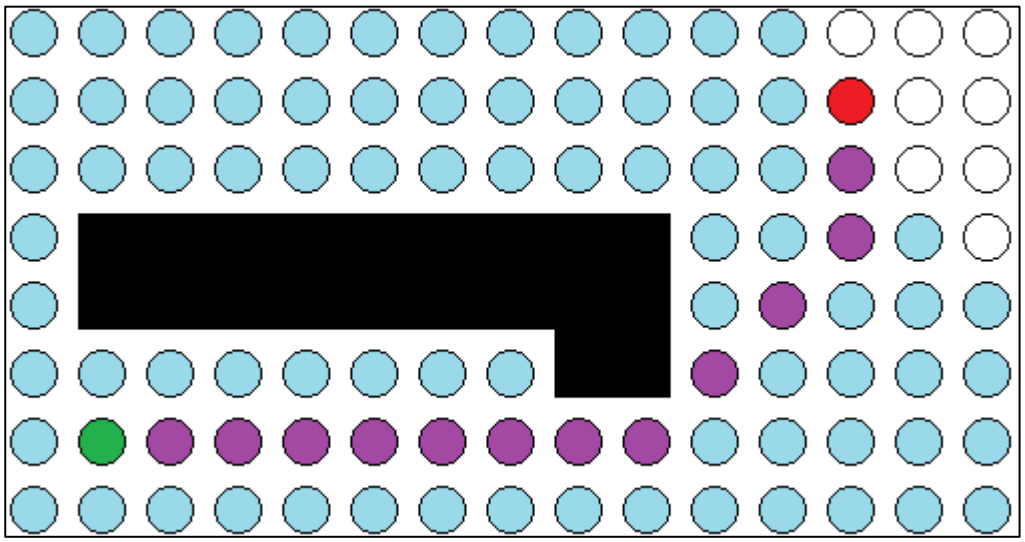

Figura 6.6 Exemple de l'aplicació de Dijkstra en un mapa de punts formant una quadrícula (2). Els nodes blaus són nodes visitats, mentre que els blancs romanen com a no visitats i els nodes liles és el camí òptim trobat. (Font pròpia)

6.1.2. Algoritme A*

L'algoritme A* té certes similituds al de Dijkstra, es pot representar el seu funcionament a partir d'un grafo com l'anterior, canvia el fet que les distàncies temptatives que proposava Dijkstra passen a ser establertes des d'un principi per valors heurístics. D'aquesta manera s'eviten avaluar nodes que no

porten enlloc i per tant aquest algoritme incrementa la rapidesa en trobar el camí òptim, també disminueix el nombre de dades utilitzades pel mateix motiu. Val a dir que el valor heurístic que se li atorga a un node ha de tenir certa coherència i precisió, per comprovar que aquests valors són fiables els creadors de l'algoritme estableixen que han de ser heurístic admissibles[1]. Si les estimacions heurístiques establertes no són correctes, és a dir no admissibles, segons Peter E. H. i els seus companys estableixen que poden ocórrer tres casos, el primer d'ells menyspreable:

"Hi ha tres casos a considerar: que l'algoritme acabi en un node que no sigui el de destí, falli en acabar i que acabi en un node de destí però sense aconseguir el cost mínim."(Peter, Nils i Raphael, 1968)

El primer cas no és possible donat que l'algoritme no acaba fins trobar-se amb el node de destí, d'altre banda tots els nodes visitats són marcats i mai més reoberts per tant sigui com sigui l'algoritme acaba quan arriba al node de destí.

$$f(n) = g(n) + h(n) \qquad \textbf{(Eq. 6.2)}$$

Aquesta equació anterior és la que defineix a l'algoritme i és la que dona un valor al node en estudi. Sent g(n) la suma de les distàncies conegudes anteriors i h(n) el valor heurístic del node en qüestió, la suma d'aquests proporciona f(n) que serà el valor que discrimini o no als demès per escollir el camí òptim.

L'algoritme A* com s'ha esmentat es basa en el que postula Dijkstra, i és d'aquesta manera que amb un mateix problema la majoria dels casos ambdós arriben a la mateixa solució. Per demostrar-ne la diferència entre ells no bastaria amb els grafos que han servit d'exemple en l'apartat anterior (6.1.1). En un camp buit de tipus graella, ambdós arriben a la mateixa solució, però A* ho fa de manera més ràpida; Amb un obstacle, es dona el mateix resultat sent encara A* el més ràpid en arribar; i així de manera consecutiva doncs mentre Dijkstra recorre tots els nodes buscant a cegues el punt de destí, A* per altre banda es recolza en l'heurística com a guia indicant la direcció o la probabilitat d'estar més a prop o lluny del destí.

[1] Es diu que un valor és heurístic admissible si aquest no sobreestima el valor de distància del node en qüestió fins al node de destí.

De manera orientativa s'ha generat una solució, per l'algoritme A*, pel mateix exemple de l'apartat anterior (Figura 6.5). Per aquest cas el resultat és el mateix ara bé, canvia en el fet de la quantitat de nodes visitats on A* se'n treu de sobre la gran majoria.

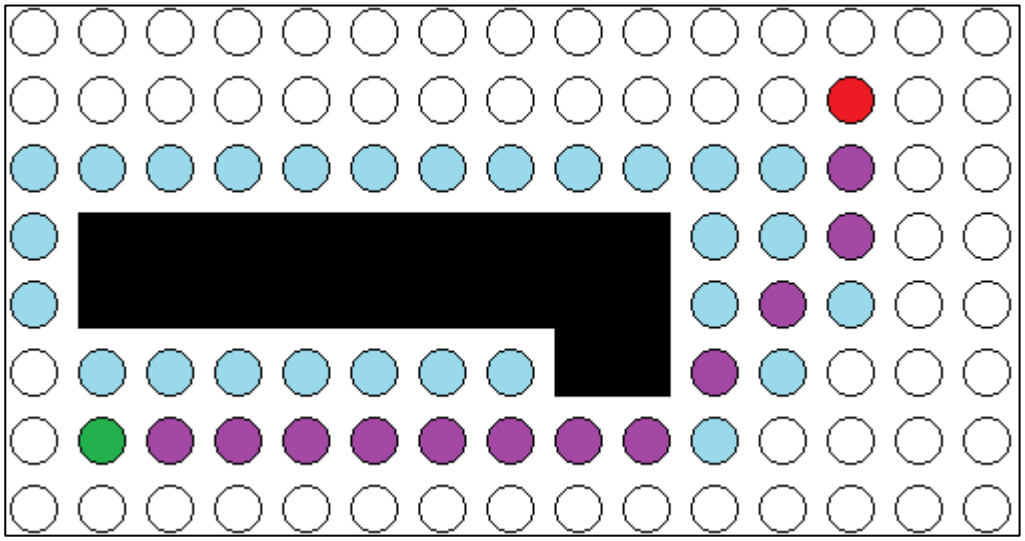

Figura 6.7 Representació de l'actuació de A* en la mateixa graella utilitzada anteriorment per Dijkstra. Verd és el node d'origen, vermell el de destí, blanc són els nodes no visitats, blau els nodes visitats i lila és el camí de nodes més curt. (Font pròpia)

6.2. Disseny i elaboració del robot i del món en 3D

Arribats a aquest punt es presenta el model elaborat per les simulacions 3D. És òptim que el model no incorpori moltes arestes per facilitar-ne el processament, per això en un principi la idea era afegir les rodes mecanum íntegrament en el robot de simulació basat en l'original (Figura 5.7), degut a la quantitat de *joints* i arestes que tenien les rodes el programa no renderitzava correctament, doncs es va desestimar la idea.

Seguint la normativa de la competició i basant-se en que el robot de la Figura 5.7 és un robot secundari, es va establir que el perímetre màxim del robot fos de 850 mm, dels quals es va decidir que el perímetre real fos d'uns 700 mm aproximadament. En definitiva el robot és una estructura de 300 mm d'alt comptant el LIDAR i de base octagonal irregular.

En aquest projecte s'han dissenyat dos robots, un d'ells és més fidel al robot original en tant a característiques, doncs és holonòmic, és fàcil d'identificar ja que no té rodes, l'altre és més fidel a l'aspecte físic i és el que té rodes. Pel que fa a termes de robòtica, holonòmic és una característica que defineix el tipus de moviments del robot referint-se a odometria. És a dir, si un robot pot moure's

endavant i endarrere i girar sobre un eix es diu que el robot no és holonòmic, mentre que si a més pot executar moviments laterals i diagonals es diu que sí ho és com en el cas de la Figura 6.8 (dreta).

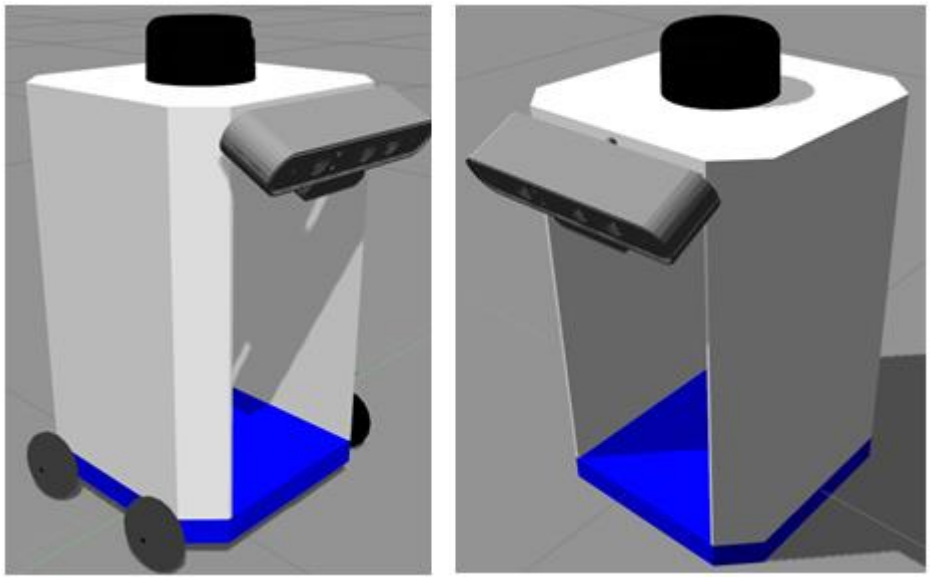

Figura 6.8 Robots creats com a models aproximats a partir del robot físic construït a l'associació. A l'esquerra el robot no holonòmic i a la dreta el robot holonòmic. (Font pròpia)

El robot original estava muntat sobre rodes mecanum que feien que el robot adquirís aquesta característica. Ara bé, com es pot observar en la Figura 6.8 (dreta), el robot no té rodes i és que com s'ha comentat a l'inici d'aquest apartat suposava massa esforç per la renderització el fet d'integrar aquestes rodes. De tal manera que el que s'ha fet és incorporar un *plugin* per poder moure el robot de manera holonòmica sense tenir rodes. Així doncs el robot de la Figura 6.8 (esquerra) no és holonòmic doncs no incorpora aquest *plugin* a més que les seves rodes no permeten tals tipus de moviments.

La raó per la qual s'incorporen dos robots en el treball s'explica més endavant en l'apartat 7.1.

La manera de dissenyar aquest robot ha sigut mitjançant FreeCad, un programa de disseny 3D gratuït i apte per les versions de Linux. Amb aquest s'han elaborat tres peces del robot: la base, el cos i les rodes del segon robot. Aquestes peces guardades en format DAE es van importar a Gazebo per comprovar-ne la correcte visualització. Un cop fet ja es podia definir el robot en format XACRO. Per evitar imprevistos es va procedir primer a construir el robot peça per peça en URDF i més endavant passar al format XACRO en tenir-ho tot enllestit.

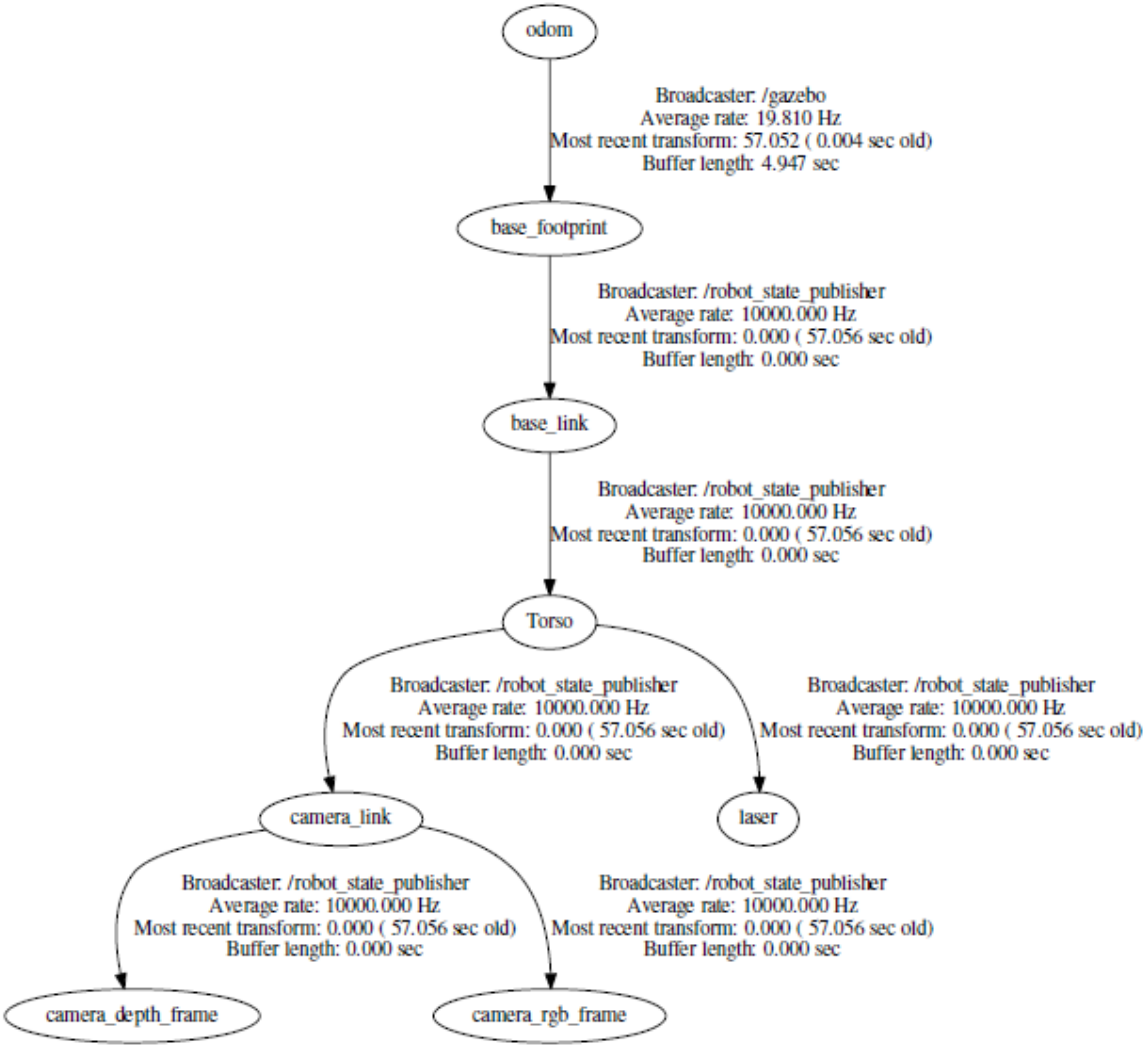

Figura 6.9 TF Tree, és l'estructura del robot en basant-se en el tòpic *transform* (*/tf*). (Font pròpia)

En la figura anterior es mostra l'esquelet del robot, aquestes són les relacions entre els *links*, els quals emmagatzemen la informació de com és la peça. En alguns casos s'han hagut de configurar alguns paràmetres d'aquests *links* com en el cas del LIDAR i la càmera, ambdós models 3D dels dispositius reals que es van intentar utilitzar.

L'altre tema a tractar és el món, el disseny 3D del camp d'Eurobot era proporcionat per la mateixa organització. A partir d'aquí només calia importar tal model a Gazebo. Ara bé, en aquest cas es va modificar via *SolidWorks* per obtenir un camp sense obstacles, i quan es diu obstacles es refereix als discs. La raó d'aquesta acció recau en el funcionament de Gazebo: a l'hora d'importar una peça, encara que aquesta sigui un assemblatge, Gazebo ho interpreta tot com una única peça, per tant l'assemblatge del camp amb totes les peces i els discos acabava sent un camp impracticable.

A partir d'aquí, tenint un camp sense discos es van crear els mons, més d'un per tenir més d'un punt d'origen i poder fer-ne les proves que calguessin amb cada un d'ells.

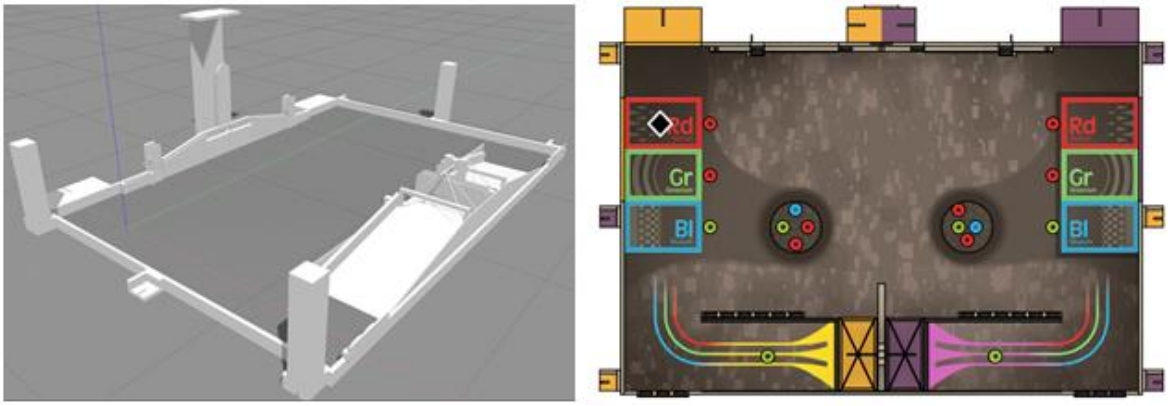

Figura 6.10 A l'esquerra el camp d'Eurobot modificat per no tenir discs i amb uns *beacons* rectangulars (de l'equip groc), en la interfície de Gazebo. A la dreta el camp d'Eurobot tret de la normativa però modificat per mostrar exactament el punt d'origen, concretament en la zona d'inici de l'equip groc. (Font pròpia)

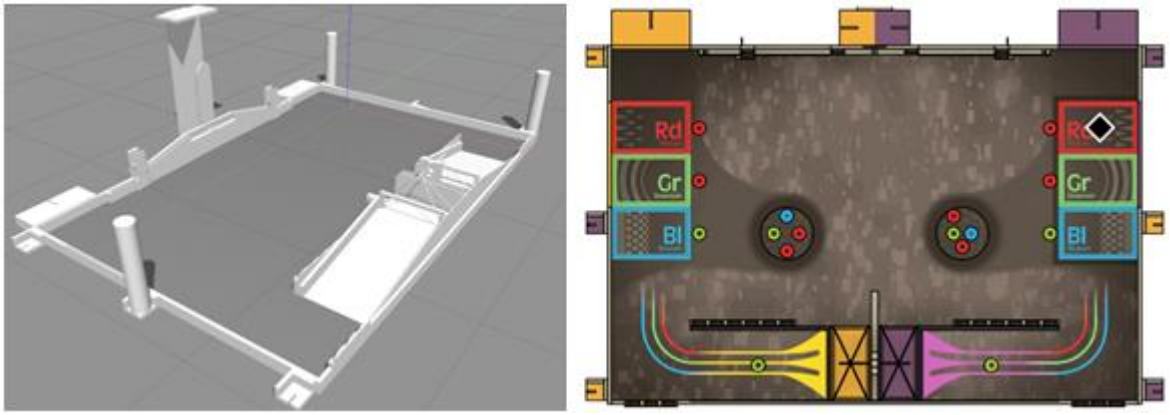

Figura 6.11 A l'esquerra el camp d'Eurobot modificat per no tenir discs i amb uns *beacons* cilíndrics (de l'equip lila), en la interfície de Gazebo. A la dreta el camp d'Eurobot tret de la normativa però modificat per mostrar exactament el punt d'origen, concretament en la zona d'inici de l'equip lila. (Font pròpia)

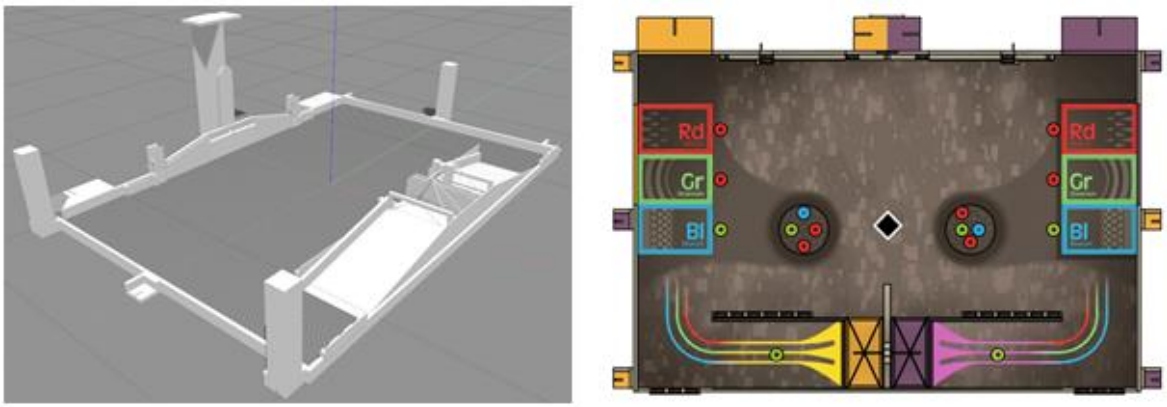

Figura 6.12 A l'esquerra el camp d'Eurobot modificat per no tenir discs i amb uns *beacons* rectangulars, en la interfície de Gazebo. A la dreta el camp d'Eurobot tret de la normativa però modificat per mostrar exactament el punt d'origen, concretament en el centre del camp. (Font pròpia)

6.3. *Navigation Stack*

ROS incorpora multitud de paquets o conjunt d'ells que permeten a l'usuari elaborar una programació detallada del robot, un d'aquests conjunts és el *Navigation Stack*. Aquest, tal i com diu el seu nom proporciona les eines necessàries per dissenyar la navegació del robot, en termes genèrics per tal de portar a terme una navegació és necessari un mapa, saber situar-se i saber traçar una ruta. Per tant, les eines que guarda aquest conjunt de paquets són la base d'aquest projecte i són el *gmapping*, el AMCL i el *move_base*. (Téllez, Ezquerro i Rodríguez, 2018)

6.3.1. Gmapping

Gmapping és el nom associat a un *package* de ROS el qual treballa amb SLAM (de les sigles en anglès *Simultaneous Localization and Mapping*), aquestes sigles defineixen a la perfecció l'acció que realitza el *Gmapping* o SLAM. En poques paraules quan un el que vol és simular el moviment d'un robot en un espai el que necessita són dues coses: un mapa (*Mapping*) i saber en tot moment on es troba el robot respecte al mapa (*Localization*). El que proporciona aquest paquet és les dues coses a l'hora, i tot i que existeixen diverses formes de localitzar-se aquest *package* opta per l'opció del làser.

Per fer-ho possible s'utilitzen els sensors del robot, i no pas el real sinó el de la simulació. Sent més concrets és necessari: un làser amb el que poder calcular distàncies, és a dir un LIDAR per exemple; i un sistema de referència que a més d'indicar el punt d'origen indiqui la relació entre aquest i tots els elements coneguts presents en l'espai, és a dir el tòpic *transform*.

Per a que tot funcioni perfectament s'ha de disposar: d'un món en el que es tingui constància d'on s'ubica l'origen, doncs serà el punt d'inici del robot; i un robot perfectament definit tenint constància d'on s'ubica el *base_link*. Lo anterior descrit és el que s'ha presentat en l'apartat anterior 6.2

Per una banda són extremament necessaris els elements mencionats per dur a terme aquest procés, però també és imprescindible configurar correctament el *package* del SLAM doncs no és genèric i s'ha d'adaptar a les condicions del robot.

Un cop tot queda establert, ja és possible activar el *gmapping* amb el robot i el món a punt en el simulador Gazebo, per fer-ho s'executa la següent comanda:

```
roslaunch rosbot_description rosbot_rviz_gmapping.launch
```

Amb aquesta comanda s'executen altres arxius *.launch* que fan aparèixer en primera instància el món i després al robot en el punt d'inici, entre d'altres arxius *.launch*. Un cop fet per poder moure el robot pel voltant, a més de que aquest incorpori els *plugins* necessaris, s'ha d'executar la següent comanda:

```
rosrun telep_tools keyboard_teleop.py
```

L'extensió .py significa python, gràcies a aquest codi python es pot controlar l'odometria i la velocitat del robot amb el teclat. Mentre es va movent el robot pel camp es va dibuixant la forma del mapa vista des del LIDAR.

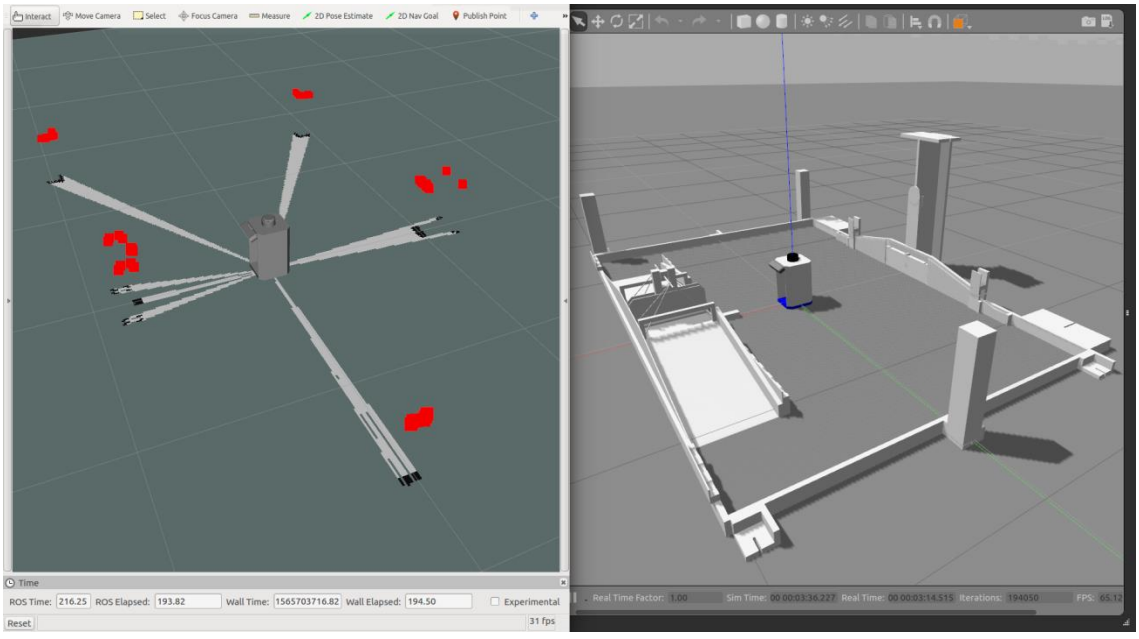

Figura 6.13 Inici del *gmapping*, a l'esquerra el robot en Rviz i a la dreta en Gazebo. (Font pròpia)

De la figura anterior cal remarcar les diferents funcions de Rviz i Gazebo, a primera vista s'observa la imatge de Rviz amb els punts vermells i les línies grises. Rviz és capaç de llegir tots els tòpics actius, durant el *gmapping* n'hi ha de referents al LIDAR, al mapa, a l'odometria, etc. i es que el que es veu com a punts vermells es precisament la visió del LIDAR. És a dir que es mostren els punts on els rajos infrarojos d'aquest dispositiu xoquen amb un obstacle, dins del seu rang de visió establert. Les línies grises, al seu torn, són les dades que s'estan generant des del robot per crear el camp: tot el que quedi registrat d'aquest color és terreny accessible, tot el que sigui de color negre és un obstacle mentre que la resta de terreny és inaccessible (depenent de la configuració del *path planning* serà accessible o no). Pel que fa a Gazebo, només representa una eina de visualització, en la que es pot observar que segueix el mateix camí que el robot fa en Rviz.

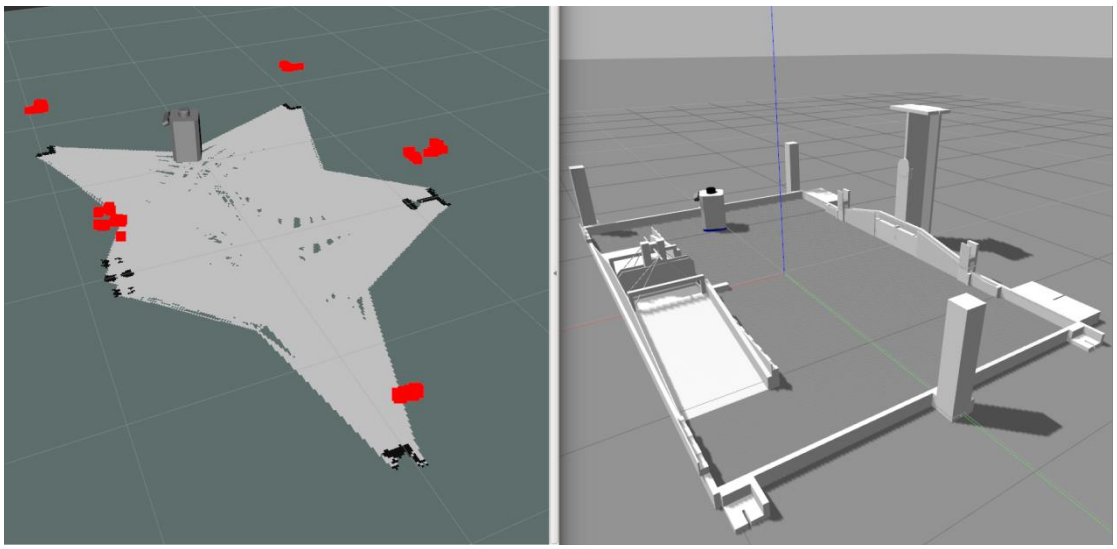

Figura 6.14 *Gmapping* en procés, a l'esquerra el robot fent el *mapping* en la interfície de Rviz i a la dreta el mateix robot en Gazebo. (Font pròpia)

Un cop escanejat tot el camp en 2D es guarda fent executar un node anomenat *map_server*. Aquest node s'encarrega bàsicament de guardar les dades que el *mapping* ha generat i després facilitar-les a qui les demani, com per exemple a AMCL. Sent més explícit, les dades que es guarden són en realitat 3 tipus de valors: 0 per terreny sense obstacles, 100 per obstacles i -1 per terreny no explorat o sense informació.

6.3.2. AMCL

AMCL de les sigles en anglès *Adaptive Monte Carlo Localization*, és un paquet que en resum tracte de localitzar el robot dins un espai conegut, en aquest cas l'espai conegut és el mapa creat pel *mapping*, i per localitzar-se utilitza en part tant les dades del sensors (LIDAR) com les dades que el *gmapping* al

seu torn ha guardat en relació al mapa creat. Per l'altre part aquest algoritme de localització utilitza un filtre de partícules basant-se en el filtre de Kalman i el mètode de localització per graelles de Markov (Dellaert *et al.*, 1999).

Més detalladament, l'algoritme de Monte Carlo (MCL) treballa amb partícules com una distribució de suposicions o possibles casos. Cada cop que el robot es mou, és una nova iteració i es genera una nova gamma de partícules, iteració rere a iteració les suposicions són contrastades amb les lectures que el robot proporciona a partir dels sensors, i cada cop com més informació es proporciona més acurades són les suposicions.

El nom de *Adaptive* es dóna pel fet que l'algoritme es pot configurar, per tant de la mateixa manera que amb el *gmapping*, depenent de les condicions del robot la configuració prendrà un caràcter o un altre. Com en el subapartat anterior es possible controlar el robot via el codi python que abans s'ha mencionat. Ara bé amb aquest *package*, que s'executa dins de Rviz es possible dirigir el robot a una posició i esperar que el càlcul de trajectòries actuï, però per a que això passi abans s'ha de configurar tant AMCL com *move_base*.

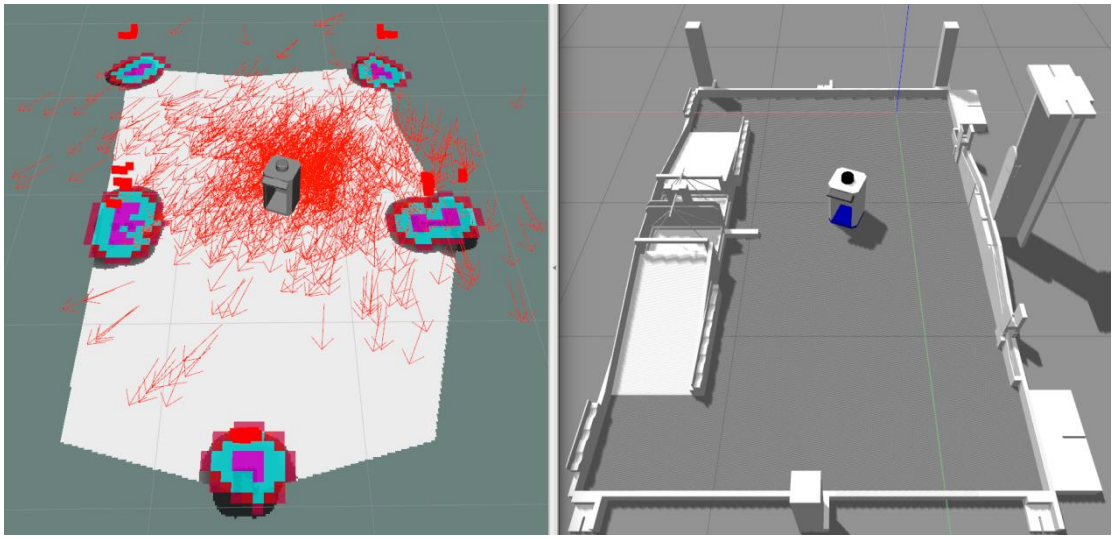

Figura 6.15 Vista del robot en els les dues interfícies de 3D amb AMCL en marxa. En vermell el conjunt de fletxes que representen les partícules o suposicions. (Font pròpia)

De la imatge anterior se'n dedueix el funcionament de AMCL, les fletxes vermelles com s'indica al peu de la imatge representen les partícules anteriorment comentades. La concentració d'aquestes s'observa que creix en un punt, en el centre normalment mentre que són disperses als extrems; i evidentment on el volum es concentra és on es troba el robot. Notis també que les fletxes indiquen la direcció.

6.3.3. Move_base i *path planning*

Move_base a més de ser un paquet és un node, i és la clau del moviment del robot. Funciona com un *SimpleActionServer* de manera que connecta altres nodes entre sí per fer funcionar la maquinària i el que és més important és l'encarregat que des d'un punt inicial el robot arribi al seu punt final prèviament indicat.

Un dels tòpics de *move_base*, move_base/goal en concret, és el que indica quin és l'objectiu o punt d'arribada en el mapa. Des de Rviz, a partir d'una eina de la que aquest programa disposa, es pot indicar de manera gràfica a on es vol que el robot vagi. A l'hora de fer-ho les coordenades del punt es guarden en el tòpic mencionat que transportarà el missatge allà on sigui requerit. Un dels destinataris és el paquet que controla el càlcul de trajectòries *nav_core*, en el qual a partir de la informació rebuda es calcula la ruta fins a la destinació assignada. Aquest tòpic, com es veurà més endavant, es pot fer servir en un codi C++ o python per assignar un nou destí al robot sense haver de fer-ho des de Rviz.

En el camí hi pot haver obstacles però és el *path planning* que evita xocar-hi. Ara bé, aquest calcula la ruta des del centre del robot sense tenir gaire en compte les dimensions d'aquest, com es pot intuir de la Figura 6.6. És per això que existeix el *costmap*, es tracte d'un mapa dels obstacles que hi ha en l'espai però amb una superfície més gran a la real, aquesta àrea de més és el cost de passar vora l'obstacle en qüestió. Aquest *costmap* utilitza el mapa anteriorment escanejat com a base i es configura en un tipus d'arxiu d'extensió YAML en el que a banda d'altres paràmetres es pot retocar el gruix del cost.

Val a dir que de *costmaps* n'hi ha dos de la mateixa manera que de *path planners*, un d'ells calcula la ruta global (*global planner*) i l'altre la ressegueix (*local planner*), mentre que pels *costmaps* un està fet pel càlcul de trajectòries global (*global costmap*) i l'altre pel local (*local costmap*).

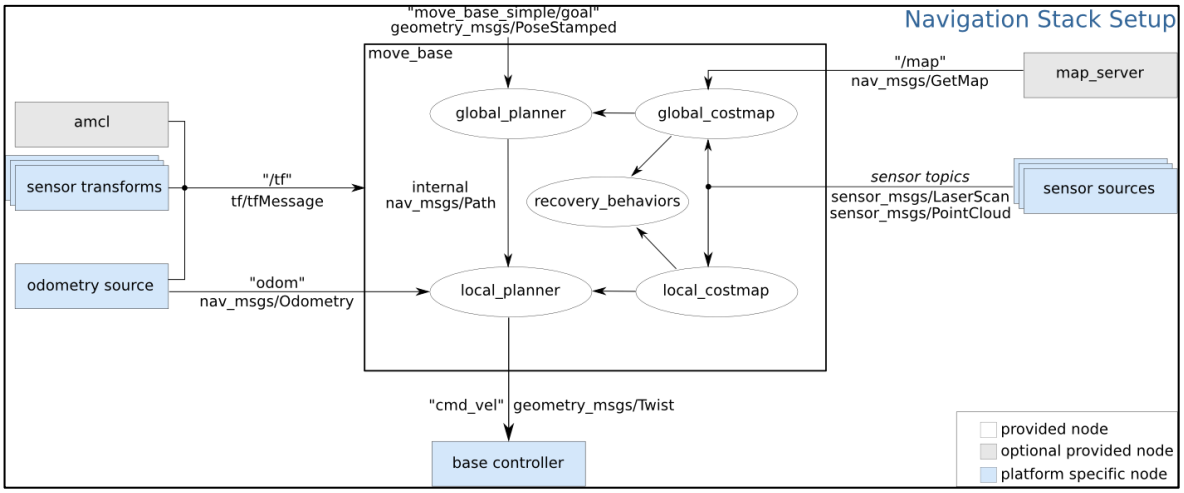

Figura 6.16 Representació gràfica de com s'estructuren el conjunt de paquets del *Navigation Stack*. (Font: (Open Source Robotic Fundation, 2011))

Com s'ha esmentat, a partir d'un punt d'inici (la posició actual del robot) i un punt de destí proporcionat per *move_base*, sempre que aquest estigui dins del mapa, el *global planner* traça una ruta entre aquests dos punts amb les dades que li proporcionen tant el *map_server* com el *global costmap*. Ara bé, per traçar aquesta ruta té diferents formes de fer-ho, o més aviat diferents algoritmes que pot fer servir, depèn de la configuració quin és l'algoritme actiu. Actualment existeixen tres algoritmes que ROS fa servir per aquest paquet i són: *navfn*, *Global Planner* i *carrot planner*. Dos d'ells ja s'han introduït abans, és a dir Dijkstra que pren el nom de *navfn* i A* que està incorporat en l'algoritme de *Global Planner*.

Navfn és íntegrament l'algoritme creat pel famós científic en computació, mentre que *Global Planner* és una barreja entre aquest algoritme i A*, i de manera que és possible configurar-ho amb diferents paràmetres com alguns del següents:

- */allow_unknow*: Paràmetre de tipus *bool* que per defecte té valor *True*, d'aquesta manera és permès que el robot navegui per zones del mapa desconegudes, és a dir que no han sigut "mapejades".
- */default_tolerance*: Aquest paràmetre de coma flotant indica en quin grau el robot pot quedar-se lluny de l'objectiu. Per defecte aquest valor és zero per tant el robot sempre s'aproparà lo màxim possible al destí.
- */visualize_potencial*: Permet veure un núvol de punts que es dibuixen en el mapa i representen els nodes visitats per l'algoritme.
- */use_dijkstra*: En valor *True* s'utilitza Dijkstra sinó s'utilitza A*.
- */use_quadratic*: En valor *True* s'utiltza una aproximació quadràtica que filtra la ruta calculada per l'algoritme de manera que queda un camí més suavitzat.

- */use_grid_path*: En valor *True* el camí que es dibuixa ressegueix els nodes del camí calculat.

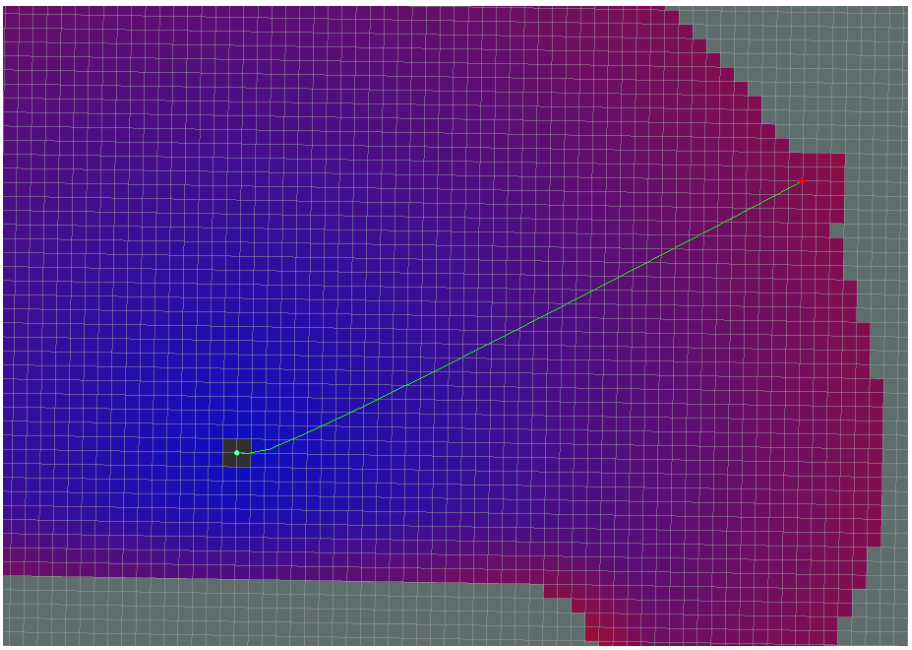

Figura 6.17 Ús de Dijkstra en el *Global Planner*, amb l'opció de *visualize_potential* activada. El punt verd és l'inici, el vermell el de destí i la línia verda és el camí calculat. Els quadrats grisos són nodes no visitats mentre que els de color són visitats. (Font: (Open Source Robotic Fundation, 2011))

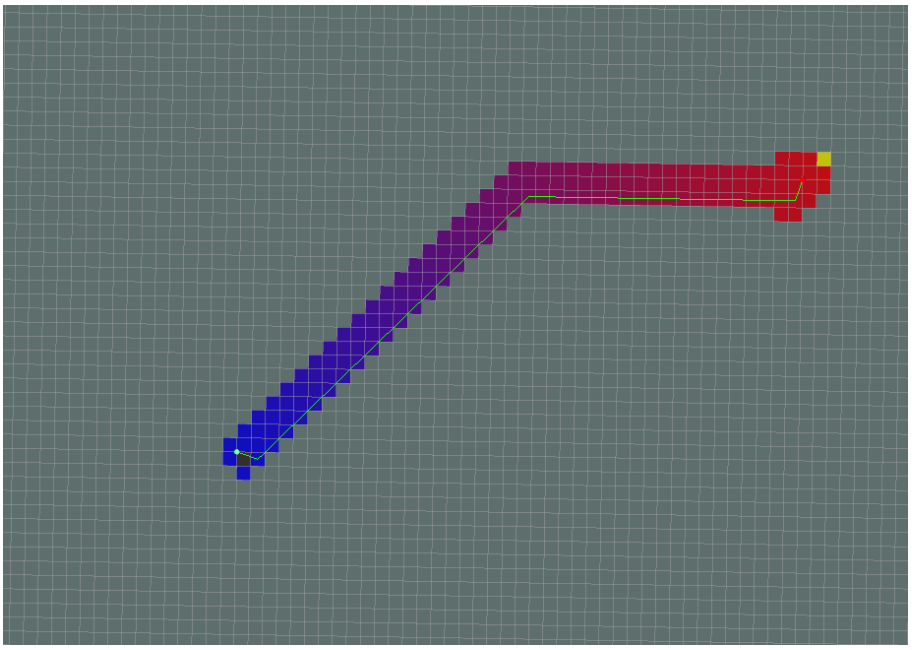

Figura 6.18 Ús de A* en el *Global Planner* amb l'opció *visualize_potential* activada. El punt verd és l'inici, el vermell el de destí i la línia verda és el camí calculat. Els quadrats grisos són nodes no visitats mentre que els de color són visitats. (Font: (Open Source Robotic Fundation, 2011))

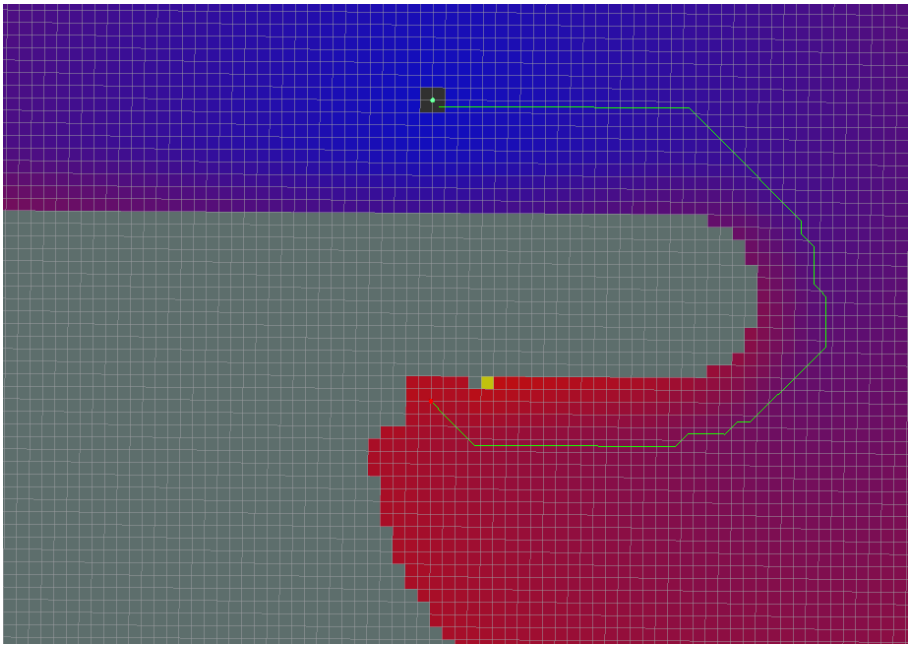

Figura 6.19 Ús de l'opció *use_grid_path*. El mateix camí calculat per qualsevol dels algoritmes utilitzats és que en resulta sense ser filtrat. (Font: (Open Source Robotic Fundation, 2011))

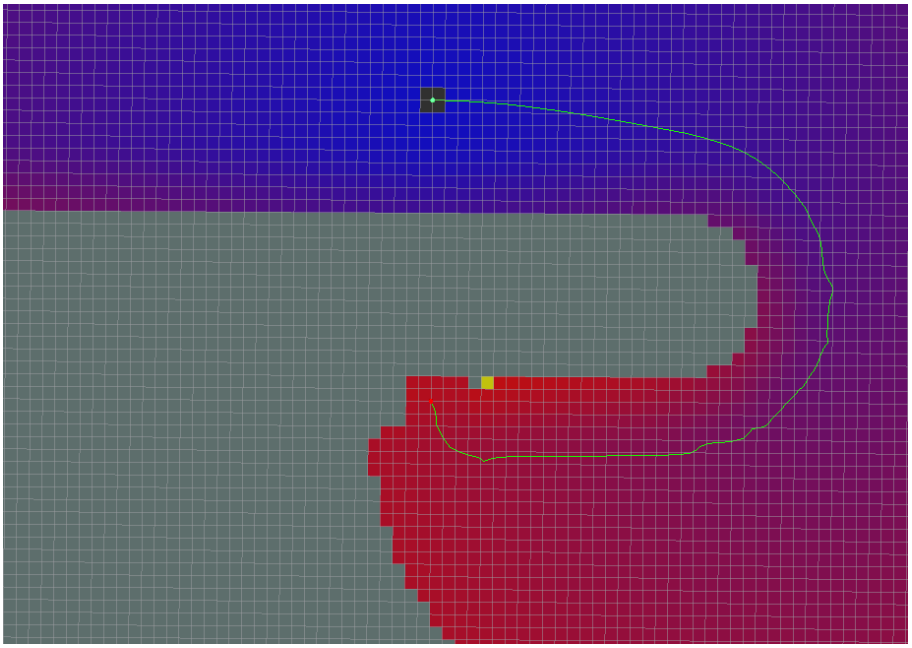

Figura 6.20 Ús de l'opció *use_quadratic*. En aquest cas el camí calculat es filtrat per una aproximació quadràtica sent aquest el resultat. (Font: (Open Source Robotic Fundation, 2011))

Com es pot observar en la Figura 6.17 i en la Figura 6.18 es demostra la diferència que suposa utilitzar un algoritme o un altre, per tant tot i que la base de A* sigui la mateixa que la de Dijkstra, el primer aporta més velocitat en el càlcul proporcionant el mateix resultat. Ara bé, aquesta comparació no és concloent doncs serien necessàries més dades per discriminar un algoritme de l'altre.

Pel que fa a *carrot planner* es tracte del més senzill entre els demès *global planners*: traça un vector directe entre el robot i el punt de destí, tot seguit comprova si aquest punt està dins d'un obstacle o no, si ho està tira enrere pel vector fins trobar un altre punt en aquest que no estigui dins d'un obstacle. I tot i que es considerat com un *global planner* podria veure's perfectament com un *local planner*, doncs argumenten R. Téllez i els seus companys:

"...aquest *planner* no fa res de *global path plannig*. És d'ajuda si necessites que el teu robot es mogui molt a prop del destí donat, encara que aquest sigui inabastable. En espais tancats i complicats, aquest *planner* no és molt pràctic. Aquest algoritme pot ser útil si, en canvi, vols que el teu robot s'apropi lo màxim possible a l'objectiu..." (Téllez, Ezquerro i Rodríguez, 2018)

I és que un cop l'algoritme troba el punt esmentat anteriorment, el passa a un *local planner* per a que sigui aquest el que actuï. De totes maneres, aquest algoritme aporta la seguretat d'arribar lo més a prop possible a un destí cosa que pot ser útil si el punt d'arribada està just en una paret o mur o una bora, com en el cas que en aquest projecte es treballa doncs el camp està envoltat de bores i hi ha objectius a sobre d'elles.

El *local planner* no només té la funció de seguir el camí que ja ha creat el *global planner* sinó també l'obligació, hom es refereix a que aquest *planner* és característic per estar calculant constantment el camí a seguir per arribar al camí establert pel càlcul global, i és que si inesperadament el robot topa amb un obstacle que no estava registrat el *local planner* té la funció d'esquivar-ho i tornar al camí marcat. Per fer-ho disposa de diversos algoritmes, diferents als ja vistos, que són: *base_local_planner*, *dwa_local_planner*, *eband_local_planner* i *teb_local_planner*.

Els dos primers són els més coneguts i més utilitzats, *base_local_planner* implementa tant l'algoritme *Dynamic Window Approach* o DWA, com el *Trajectory Rollout*. Tal algoritme, el DWA, es basa en una cerca dins d'un espai de velocitats, les quals depenen de les restriccions que imposa la dinàmica de robot. El grup de cerca es redueix quan només es consideren, entre totes, les velocitats segures respecte als obstacles. D'aquest grup finalment se'n treu una d'escollida que és la que maximitza la funció de l'algoritme respecte a l'objectiu. (Fox, Burgad i Thrun, 1997)

Per altre banda, tot i que *Trajectory Rollout* es basa també en la cerca de velocitats òptimes ho fa amb un espai de cerca diferent.

Dwa_local_planner implementa únicament l'algoritme que anteriorment s'ha mencionat i ve a ser el mateix però amb diferències en la codificació, fent-la més aclaridora.

I finalment tant *eband_local_planner* com *teb_local_planner* són paquets que incorporen el mètode de la Banda Elàstica o *Elastic Band*.

6.4. App python, *StrategyLoop*

En aquest apartat es dona a conèixer l'aplicació que s'ha dut a terme amb python. Es tracte d'un node, anomenat *StrategyLoop* creat expressament per monitoritzar millor l'estratègia, fent anar al robot a punts concrets del mapa d'Eurobot prèviament indicats, a més es dona l'opció d'indicar una seqüència de punts de tal manera que el robot la ressegueixi movent-se pel mapa utilitzant el càlcul de trajectòries per arribar als punts de destí.

Una altre raó per la qual s'ha realitzat aquesta aplicació és per la manca d'obstacles que s'han registrat amb el *gmapping* en el mapa d'Eurobot. Sent més aclaridor, en la Figura 6.21 s'indiquen en verd diferents zones on el robot hi podria o voldria accedir-hi. Ara bé, a causa de l'obstacle que la mateixa estructura del camp suposa, el robot hauria d'evitar-ho depenent del punt d'inici. El problema està en la posició del LIDAR que la normativa de la competició imposa, d'aquesta manera a l'hora de realitzar el escanejat queda una imatge com la que es mostra en la Figura 6.22. Evidentment amb aquesta imatge com a mapa, el càlcul de trajectòries no pot funcionar a la perfecció doncs suposaria un xoc amb l'estructura el fet de voler situar el robot en alguna de les zones mencionades.

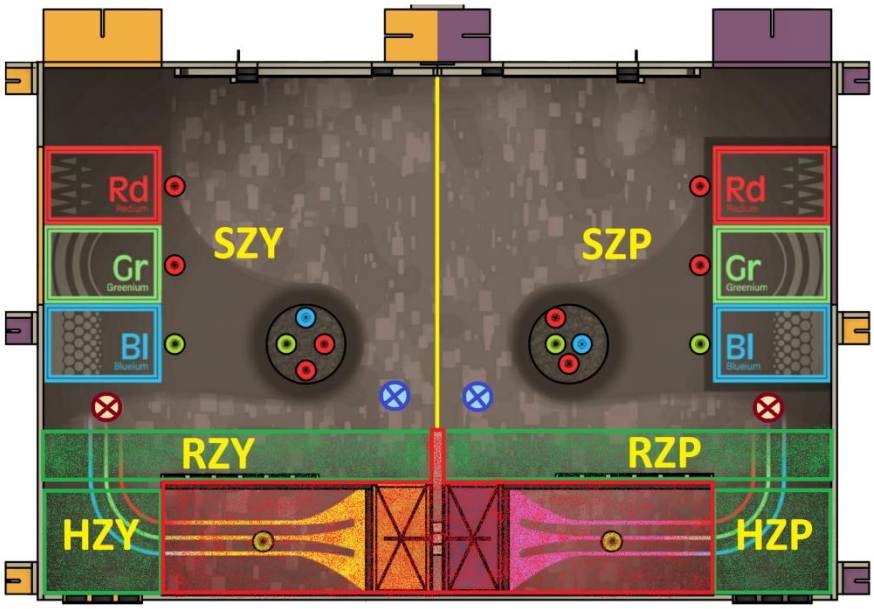

Figura 6.21 Camp d'Eurobot tret de la normativa, modificat per representar les diferents zones útils per l'aplicació. En verd les zones on el robot pot tenir problemes per accedir depenent del punt de partida, en vermell les zones considerades obstacles de la mateixa estructura del camp. Els cercles amb creus són punts intermedis o punts d'accés a les zones verdes, els blaus per les zones de risc (RZY, RZP) i els vermells per les zones perilloses (HZY, HZP). (Font pròpia)

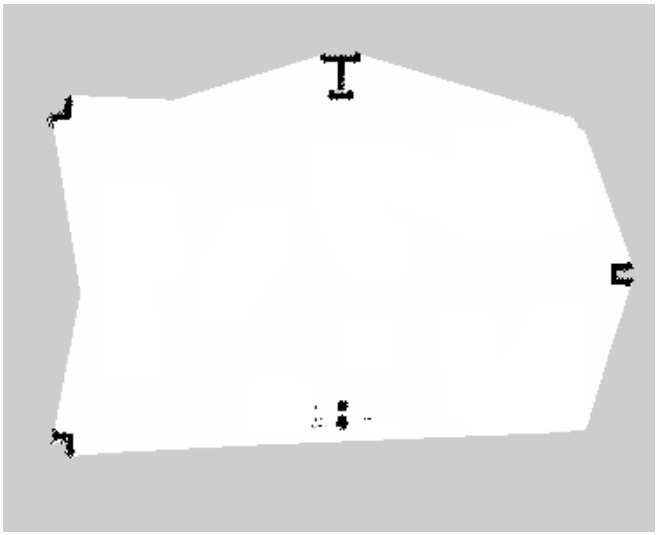

Figura 6.22 Mapa creat en el *gmapping*, on s'hi observa la manca d'obstacles. (Font pròpia)

Per tant per solucionar-ho es va idear aquest node, que la seva funció principal és publicar en el tòpic *move_base/goal* per indicar al robot quin és el pròxim destí, i depenent quin sigui es tracen uns altres punts intermedis per a que el robot hi passi abans d'arribar al punt final. De punts intermedis exactament hi ha quatre en tot el camp, són els indicats en la figura anterior (Figura 6.21). A més en aquesta figura anteriorment indicada s´hi mostren altres zones marcades amb títols que són:

- SZY (*Safe Zone Yellow*)
- SZP (*Safe Zone Purple*)
- RZY (*Risk Zone Yellow*)
- RZP (*Risk Zone Purple*)
- HZY (*Hazard Zone Yellow*)
- HZP (*Hazard Zone Purple*)

Segons la zona de destí i la zona d'origen, l'algoritme de l'aplicació aplica més o menys punts intermedis per tal de fer arribar el robot sense que col·lisioni amb cap obstacle estructural.

Com s'ha esmentat anteriorment, */move_base* treballa sobre un *ActionServer* controlant diversos paquets de dades, per tant s'ha hagut de crear un *ActionClient* per poder comunicar el tòpic sobre el qual es volia escriure, és estrictament necessari que el node AMC L estigui en marxa per fer funcionar aquest node. El funcionament específic del codi s'explica a continuació.

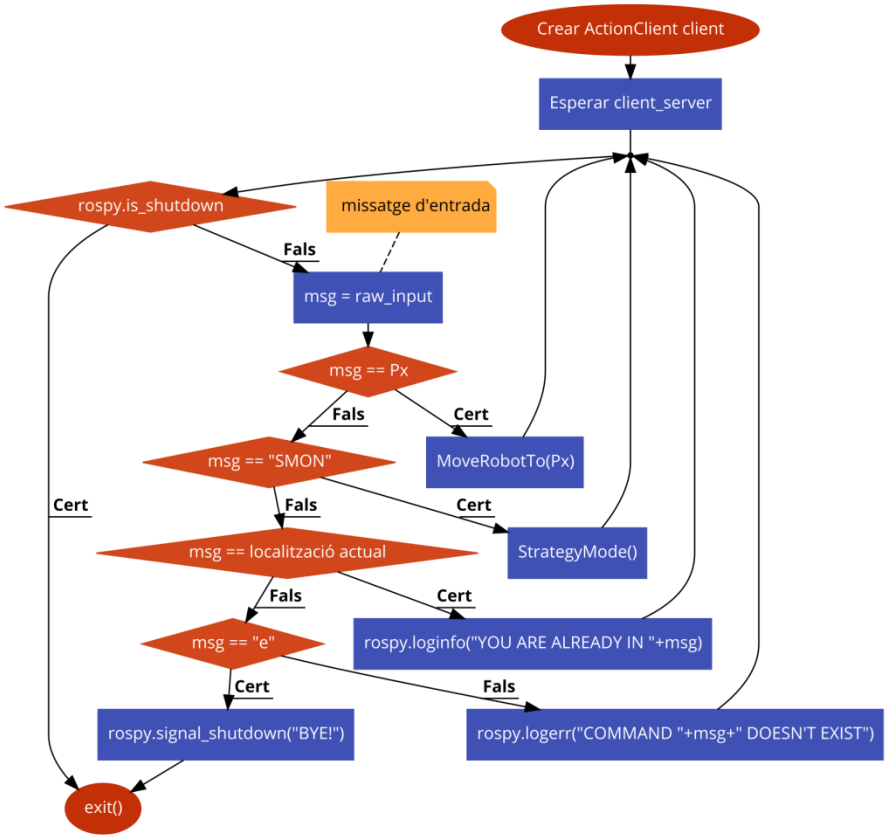

Figura 6.23 Representació del codi en un gràfic de flux, concretament es tracte de la inicialització i el bucle principal. (Font pròpia)

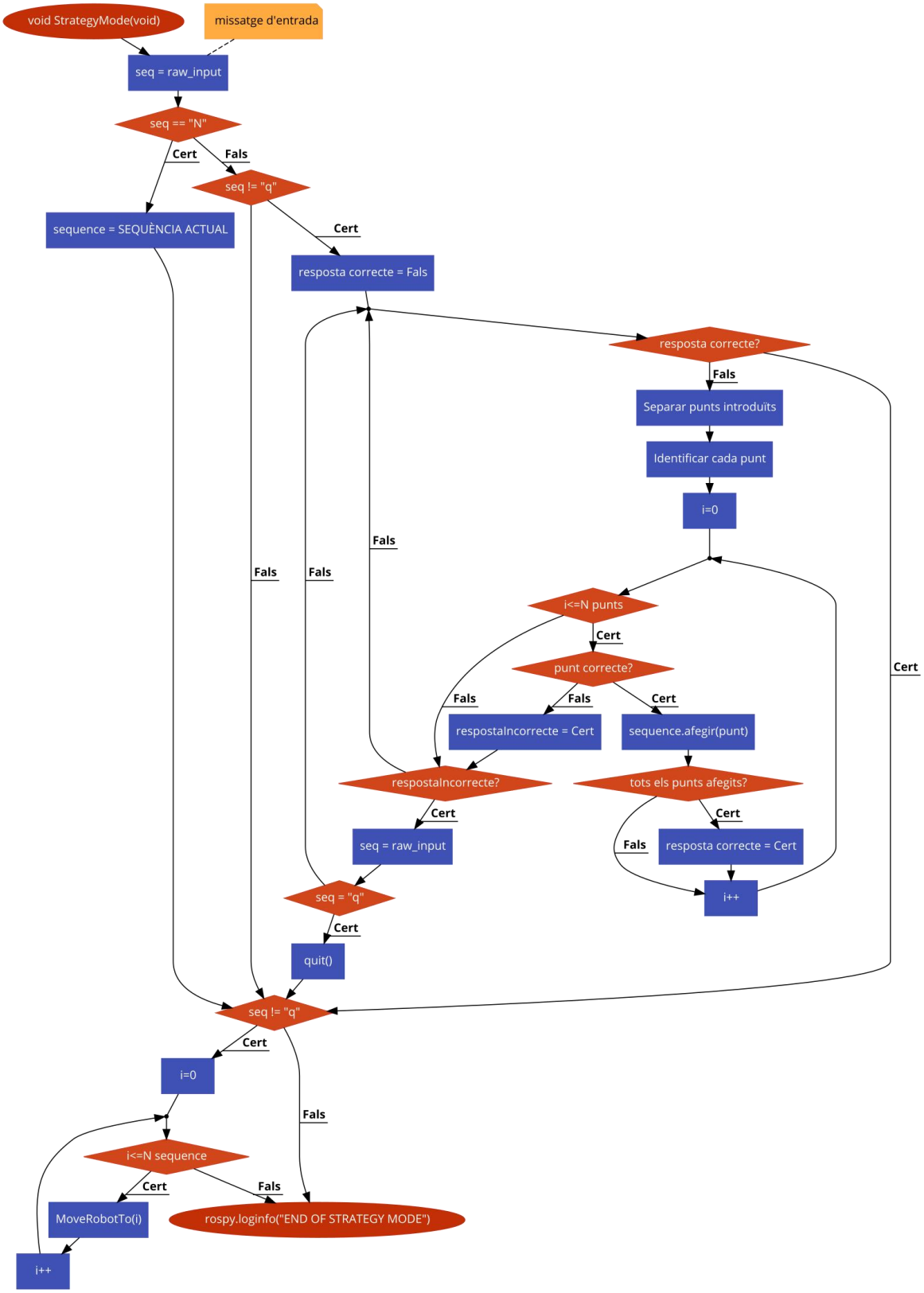

Figura 6.24 Representació del codi en un gràfic de flux, concretament es tracte de la funció *StrategyMode*. (Font pròpia)

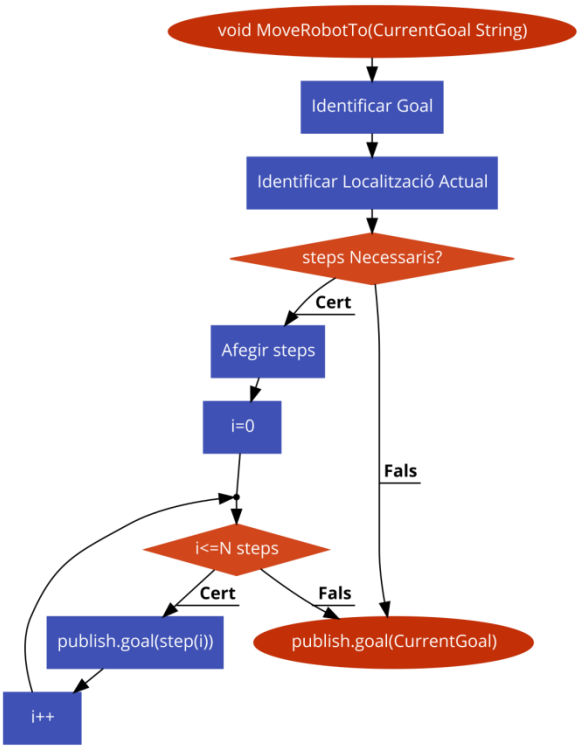

Figura 6.25 Representació del codi en un gràfic de flux, concretament es tracte de la funció *MoveRobotTo*. (Font pròpia)

El programa treballa amb l'usuari per la finestra de comandes, així que un cop iniciat es demana que s'introdueixi una comanda. En cas de detectar-ne una de desconeguda el programa avisa i torna a preguntar per introduir una ordre, entre les comandes acceptades estan les següents:

- SMON (*Strategy Mode ON*): Quan s'inicia la app per defecte aquest mode està apagat, per defecte l'usuari pot indicar al robot anar a un punt en concret però no pot introduir una seqüència. Per fer-ho ha d'entrar en aquest mode.
- P1, P2, P3, …, P12: Són els punts que amb el mode d'estratègia desactivat es pot indicar al robot per anar-hi. Aquests punts són els indicats en la Figura 6.26.
- E o e (*Exit*): En el mode per defecte s'utilitza aquesta comanda per sortir de l'aplicació.
- Q o q (*Quit*): En el mode SMON s'utilitza aquesta comanda per sortir del mode.
- 1.5.8.6.7: És un exemple de com introduir una seqüència de punts dels mapa, separats per punts, si un dels punts no està registrat el programa ho detecta i avisa de que la seqüència és incorrecta.

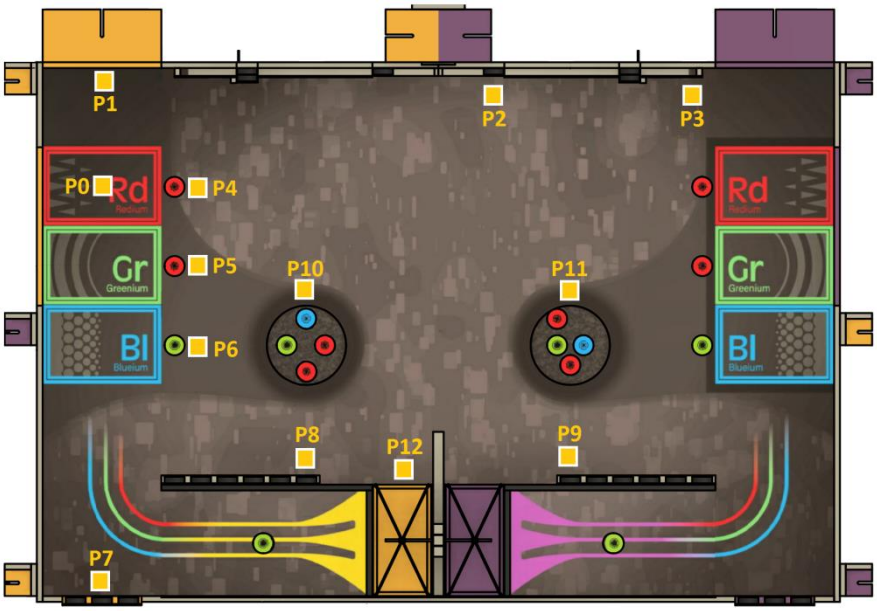

Figura 6.26 Camp d'Eurobot tret de la normativa, modificat per representar els punts establerts en l'aplicació. La majoria d'ells són d'ús preferent pel grup groc. (Font pròpia)

7. Protocol de proves i Avaluació de riscos

En aquest apartat l'objectiu es l'estudi del progrés del *package* del projecte, per tant es tenen en compte tots els canvis fets que hagin pogut provocar una variació en el comportament del robot vers el que es volia aconseguir des d'un principi. Val a dir que aquest projecte està basat en un *package* ja existent com s'ha esmentat en la introducció, d'aquesta manera molts dels paràmetres i arxius no han sigut definits o creats des de zero, ara bé la gran majoria s'han hagut de modificar per a que complissin amb els requisits que hom es proposava.

7.1. Procés de configuració del robot

Inicialment el *package* incorpora ja un robot, es tracte de Rosbot: un robot no holonòmic i de petites dimensions que incorpora un LIDAR i una càmera de profunditat. Rosbot es mou per l'espai gràcies a les seves quatre rodes lligades cada una a una *joint*, les quals són controlables a partir del *plugin differential drive*; a més incorpora quatre sensors d'infraroig per detectar les proximitats. Tots aquests elements queden reflectits en els arxius XACRO que defineixen el robot. En total en el *package* original n'hi ha quatre arxius: el primer (macros.xacro) descriu multitud d'equacions que defineixen les inèrcies de diferents cossos i parts del robot, però no té utilitat ja que aquest arxiu esta completament comentat; el segon (materials.xacro) si que té utilitat però és irrellevant ja que només defineix els colors del robot; per altre banda els dos últims són els més importants, un d'ells (rosbot.gazebo) només defineix tots els *plugins* que el robot necessita, mentre que l'altre (rosbot.xacro) defineix l'estructura del robot, és a dir *links* i *joints*.

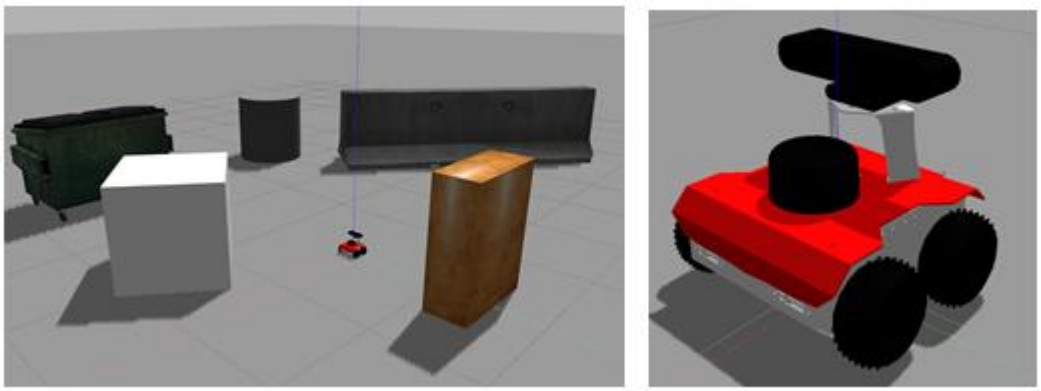

Figura 7.1 Rosbot a la dreta. A l'esquerra el mateix robot en el mapa predeterminat de *Turtlebot*. (Font pròpia)

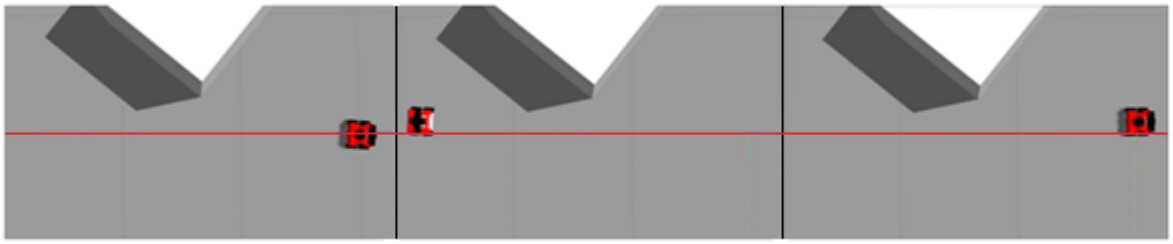

Figura 7.2 Rosbot vist des de d'alt en Gazebo conduït per teclat. Es mostra una certa desviació. (Font pròpia)

Un dels principals problemes, o característica, que es va trobar en el robot era una certa desviació en la conducció d'aquest. Podria tractar-se d'una característica del mateix *package* que pretén aproximar a la realitat el moviment del robot, no s'ha trobat cap referencia al respecte. De totes maneres suposa un problema en qualsevol situació doncs un robot amb desviació tant en simulació com en la realitat suposa una acumulació d'errors, per tant és un dels aspectes a corregir.

Aquesta és la primera part del projecte, crear un model 3D per utilitzar-ho com ha robot. Un cop fet via FreeCad s'assemblen totes les peces en un arxiu URDF i així fins arribar al robot de la Figura 6.8. En primera instància la principal diferència amb el robot original són les rodes, es necessitava un robot capaç de fer moviments laterals i diagonals; donat que no era possible assemblar les rodes mecanum utilitzades en el robot real es va decidir utilitzar un *plugin* especial anomenat *Planar move*, el qual permet moviments holonòmics amb l'inconvenient de no utilitzar rodes. És un inconvenient pel fet que el robot ha d'estar sempre horitzontal, d'aquesta manera no és possible pujar la rampa del camp d'Eurobot. Per altre banda no és un problema per que no és obligatori pujar la rampa per utilitzar la bàscula d'àtoms, hi ha un altre accés al que la normativa no fa referència de no fer servir.

Per aquesta raó i per prevenir en cas de no trobar solució al problema del tipus de moviments del robot es va crear un segon robot amb rodes i no holonòmic, que fins ara no s'ha hagut de fer servir.

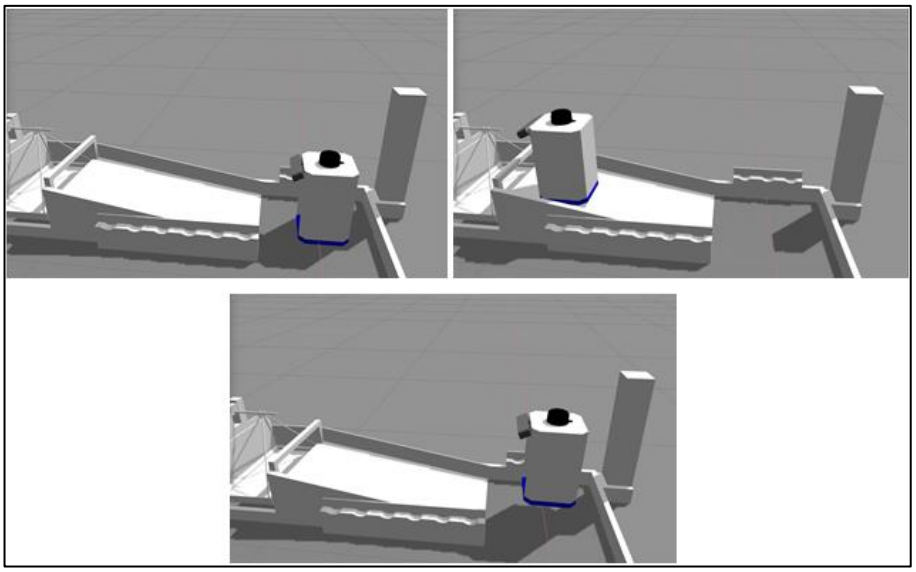

Figura 7.3 Robot amb el *plugin Planar move* pujant la rampa del camp d'Eurobot. (Font pròpia)

Fent tests dels moviments del robot en el camp hom es troba amb un problema causat aparentment per les físiques o bé del camp o bé del robot. En la figura següent se'n mostra la problemàtica.

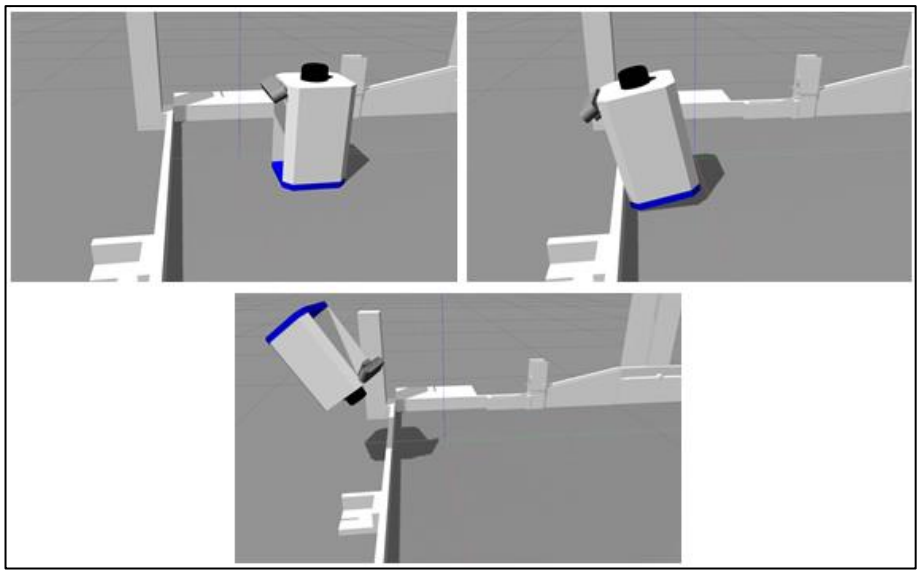

Figura 7.4 Problemàtica amb les físiques del robot. (Font pròpia)

En la imatge anterior s'observa que el robot a l'hora d'entrar en contacte amb el mur del camp, aquest li causa una força que el fa primer inclinar-se i després de sobte experimenta una expulsió que el fa aixecar-se del terra i rotar en varies direccions de forma indefinida. En aquest moment un pot pensar que el problema s'origina per la posició massa elevada del centre de massa del robot. Ara bé, les condicions d'utilitzar el *plugin Planar move* són:

"Nota: L'objecte necessita tenir suficient inèrcia per prevenir moviments no desitjats – els quals poden ocórrer com a reacció de la velocitat entregada. Pots intentar-ho incrementant la inèrcia fins que l'objecte es mogui com es desitja. També és bo tenir un centre de massa proper al terra." (Open Source Robotics Foundation, 2014)

Per tant, hom dedueix que el robot esta experimentant "moviments no desitjats", de tal manera que cal augmentar la inèrcia i fer baixar el centre de massa. Per disminuir la posició del centre de massa només cal que la base sigui més pesada que el cos, de totes maneres des d'un inici aquesta condició ja es complia sent la base de 10 Kg i el cos de 5 Kg (no es tenen en compte els altres objectes com la càmera i el LIDAR). Tenint unes inèrcies de 0,08 Kg/m^2 i 0,06 Kg/m^2 respectivament es decideix fer un segon intent.

		Intent 0	Intent 1	Intent 2	Intent 3
Base	Massa (Kg)	10,00	10,00	10,00	40,00
	Inèrcia (Kg/m^2)	0,08	1,00	10,00	40,00
Cos	Massa (Kg)	5,00	5,00	5,00	40,00
	Inèrcia (Kg/m^2)	0,06	1,00	10,00	40,00
Resultat					

Taula 7.1 Representació dels intents segons la modificació de la massa i/o la inèrcia del robot, amb els respectius resultats depenent si persisteix el problema de les físiques o no. (Font pròpia)

Intent rere intent s'ha anat augmentat el valor de la inèrcia fins arribar a valors desmesurats, no obstant el resultat obtingut continua sent el mateix. Val a dir que tot i que en l'últim intent els valors de massa i inèrcia siguin tots iguals el centre de massa continua essent a prop del terra. Arribats a aquest punt cal canviar el punt de vista del problema: el robot experimenta un moviment inesperat a l'hora de xocar frontalment amb el mur del camp, però ho fa quan xoca de manera lateral? Per comprovar-ho es tornen a establir els valors de l'intent 1.

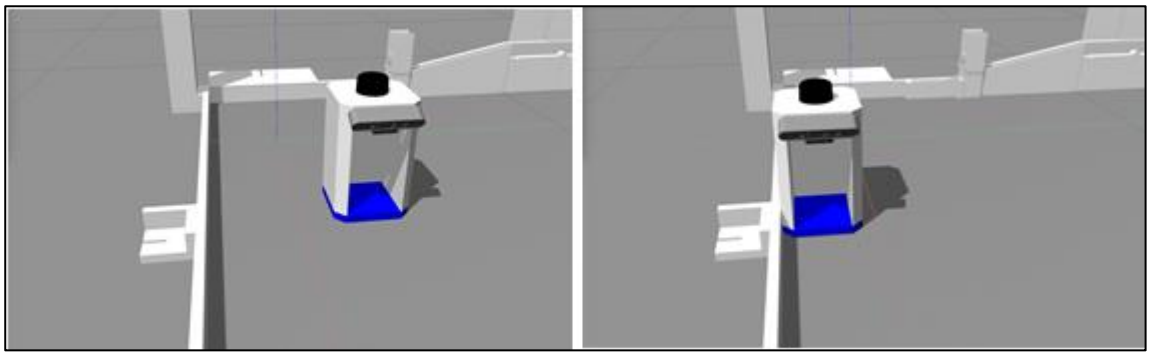

Figura 7.5 Xoc lateral del robot amb el mur del camp d'Eurobot, no es detecta el problema inicial. (Font pròpia)

Per tant, tot i que el problema sembla provenir de la inèrcia no té gaire sentit físic el que s'observa entre les Figures Figura 7.4 i Figura 7.5. No obstant, l'explicació que hom pot donar es la diferència entre els dos casos: en el primer cas el robot xoca amb la part buida del cos mentre que en el segon cas ho fa amb una de les parets llises. D'aquí l'única conclusió que se'n pot extreure és que hi ha algun element del robot que afecta a la seva física i que és diferent en la part frontal respecte a la lateral; deixant de banda les masses i les inèrcies l'únic element d'aquestes característiques descrites és la *collision*. Aquesta, com es mostra en la Figura 4.8 representa la forma geomètrica que realment xoca amb la resta d'elements en l'espai, lo habitual és pensar que ha de tenir una forma idèntica a la part visual però es recomana que sigui d'una geometria bàsica per evitar casos com aquests.

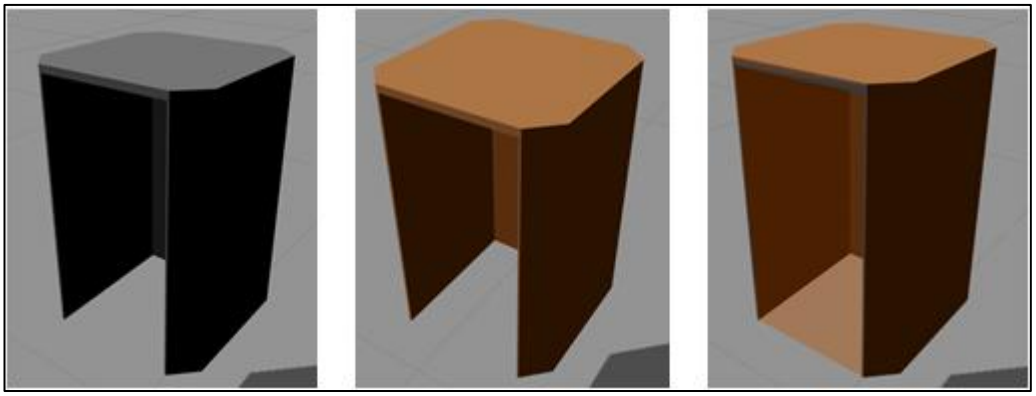

Figura 7.6 A l'esquerra el cos del robot sense visualitzar la *collision*. Al centre el cos del robot visualitzant en taronja la *collision* inicial. A la dreta el cos del robot visualitzant en taronja la *collision* modificada. (Font pròpia)

Per tant el següent intent comença per modificar la forma geomètrica de la *collision*, la qual passa simplement a no incorporar el buit del robot original com es mostra en la figura anterior.

Un cop fetes les proves pertinents el resultat és favorable, és a dir que s'ha trobat definitivament la font del problema i s'ha aconseguit solucionar. Així que finalment queda configurat el robot.

7.2. Parametrització del *gmapping*

En l'inici del projecte l'arxiu referent als paràmetres de *gmapping* hi era present però amb molts menys paràmetres dels que es pot arribar a definir, de la mateixa manera que amb AMCL tal i com s'explica en l'apartat corresponent. De totes maneres a l'hora d'utilitzar el *package* original, tot i que certs aspectes eren millorables de per sí, el funcionament d'aquest era correcte.

Turtlebot és un altre *package* totalment construït que incorpora les mateixes funcionalitats, però a diferència del qual s'ha partit el primer no té res a envejar del segon. En quant a *gmapping*, i també

AMCL, els arxius de paràmetres d'aquests són molt més complerts, per tant es va decidir basar-se en ells com a guia.

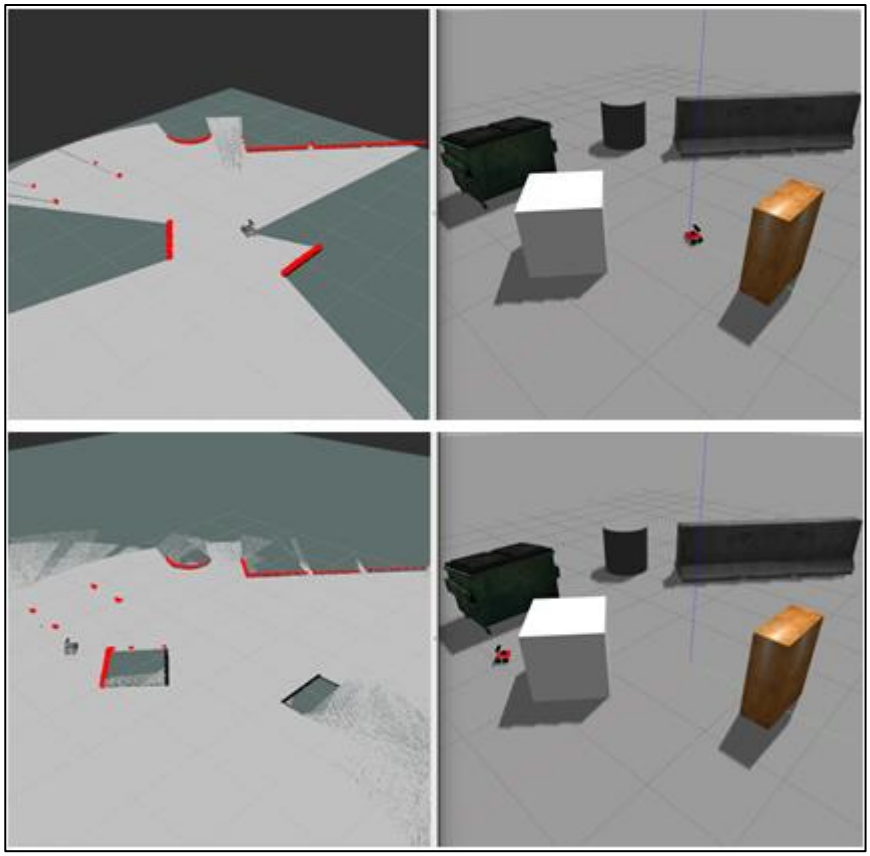

Figura 7.7 *Gmapping* amb el robot original en el món predeterminat de *Turtlebot*. (Font pròpia)

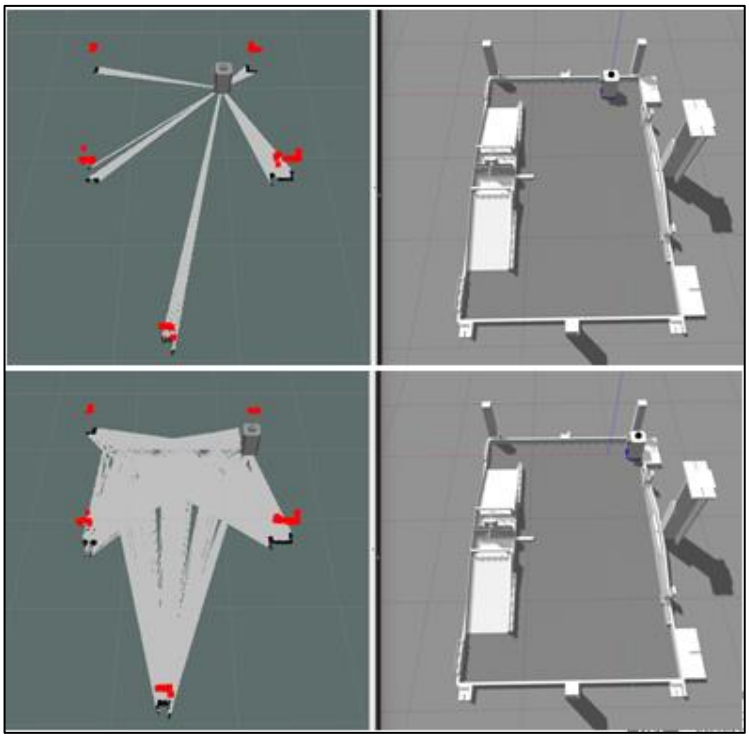

Figura 7.8 *Gmapping* amb la configuració original, robot nou i món d'Eurobot. (Font pròpia)

En un principi no es nota cap irregularitat en el funcionament del *gmapping* en el paquet original, hi ha certs aspectes que es podrien millorar com per exemple els intervals de càrrega de dades entre d'altres. Tot i així la diferència no seria molt gran.

Paràmetres originals	valor	Paràmetres *Turtlebot*	valor
base_frame	base_link	base_frame	base_link
odom_frame	odom	odom_frame	odom
map_update_interval	1	map_update_interval	5
maxUrange	5	maxUrange	3,5
		maxRange	8
		sigma	0,05
		kernelSize	1
		lstep	0,05
		astep	0,05
		iterations	5
		lsigma	0,075
		ogain	3
		lskip	0
		minimumScore	200
		srr	0,01
		srt	0,02
		str	0,01
		stt	0,02
linearUpdate	0,05	linearUpdate	0,5
angularUpdate	0,05	angularUpdate	0,436
temporalUpdate	0,1	temporalUpdate	-1
		resampleThreshold	0,5
particles	100	particles	80
xmin	-5	xmin	-1
ymin	-5	ymin	-1
xmax	5	xmax	1
ymax	5	ymax	1
delta	0,01	delta	0,05
		llsamplerange	0,01
		llsamplestep	0,01
		lasmplerange	0,005
		lasamplestep	0,005

Taula 7.2 Comparació entre els paràmetres originals i els de *Turtlebot* del *gmapping*. En vermell els valors antics, en verd els valors i paràmetres nous, i en groc els paràmetres amb els valors per defecte. (Font pròpia)

Es pot mal interpretar que un paràmetre en el seu valor per defecte és el mateix que no declarar-lo, en canvi és tot el contrari. Un paràmetre no declarat és inexistent per tant no es considera en l'algoritme. Entre tots els paràmetres els més destacats podrien ser:

- *base_frame*. No tant important com *odom_frame*, però per darrere doncs és essencial lligar aquest paràmetre amb la base del robot, en el cas d'aquest projecte la base s'anomena *base_footprint*, que ve a ser un altre nom usual en la nomenclatura de ROS.
- *odom_frame.* Aquest cal lligar-lo al tòpic referent a la odometria, tot hi que existeixin ja alguns tòpic com aquest, és possible que se'n canviï el nom amb la comanda *remap*. És important que el lligam sigui el correcte, en cas contrari *tf* pot deixar de funcionar correctament i és possible que es desvinculin els elements en l'espai o simplement que *gmapping* no funcioni correctament.
- *maxUrange*. S'estableix aquest paràmetre com el màxim valor útil del rang del sensor, en aquest cas 3,5 m doncs el camp com a màxim mesura 3 metres, es deixa un marge de mig metre.

En el cas de no haver obstacles en el rang de visió del sensor i es vol que aquest espai aparegui com espai lliure s'ha de seguir la següent regla.

maxUrange < rang màxim del sensor real <= *maxRange* **(Eq. 7.1)**

El camp és de 2 m per 3 m i rere els obstacles que el LIDAR pugui veure en els límits d'aquest no interessa que aparegui l'espai com a lliure sinó com a inexplorat, per tant el valor de *maxUrange* ha de ser menor al valor màxim del rang del sensor mentre que *maxRange* és el valor màxim real, és a dir 8 m.

- *maxRange.* S'estableix a 8 m seguint les indicacions anteriors.
- *minimumScore*. Es tracte d'un valor llindar que analitzant la sortida de dades del *scan* es decideix si és una bona lectura (per sota del valor) o no (per sobre del valor). A efectes pràctics és un paràmetre capaç d'evitar salts en l'estimació del robot quan aquest pretén realitzar la part de localització del SLAM. Els resultats d'aquest paràmetre són més evidents quan el robot incorpora un sensor de rang limitat (de 5 m per exemple); el valor per defecte és 0 per tant es manté en 200
- *linearUpdate.* Aquest i els dos següents són paràmetres que afecten als intervals de càrrega de dades, en concret *linearUpdate* indica cada quanta distància s'ha de fer un procés de *scan* i es representa en metres.
- *angularUpdate.* Indica cada quants graus de gir s'ha de fer un procés de *scan*.
- *temporalUpdate.* Indica cada quants segons s'ha de fer un procés de *scan*, sempre en el cas que l'últim fet sigui més antic que el valor de temps establert.
- *xmax*, *ymax*, *ymin*, *xmin.* En metres s'indica la grandària del camp.
- *delta*. És la relació entre el nombre de metres per graella, és a dir la resolució del mapa.

Paràmetres *Turtlebot*	valor	Paràmetres finals	valor
base_frame	base_link	base_frame	base_footprint
odom_frame	odom	odom_frame	odom
map_update_interval	5	map_update_interval	1
maxUrange	6	maxUrange	3,5
maxRange	8	maxRange	8
sigma	0,05	sigma	0,05
kernelSize	1	kernelSize	1
lstep	0,05	lstep	0,05
astep	0,05	astep	0,05
iterations	5	iterations	5
lsigma	0,075	lsigma	0,075
ogain	3	ogain	3
lskip	0	lskip	0
minimumScore	200	minimumScore	200
srr	0,01	srr	0,01
srt	0,02	srt	0,02
str	0,01	str	0,01
stt	0,02	stt	0,02
linearUpdate	0,5	linearUpdate	0,05
angularUpdate	0,436	angularUpdate	0,05
temporalUpdate	-1	temporalUpdate	0,7
resampleThreshold	0,5	resampleThreshold	0,5
particles	80	particles	80
xmin	-1	xmin	-5
ymin	-1	ymin	-5
xmax	1	xmax	5
ymax	1	ymax	5
delta	0,05	delta	0,01
llsamplerange	0,01	llsamplerange	0,01
llsamplestep	0,01	llsamplestep	0,01
lasmplerange	0,005	lasmplerange	0,005
lasamplestep	0,005	lasamplestep	0,005

Taula 7.3 Comparació entre els paràmetres de *Turtlebot* i els finalment adoptats. En vermell els valors antics i en verd els valors nous. (Font pròpia)

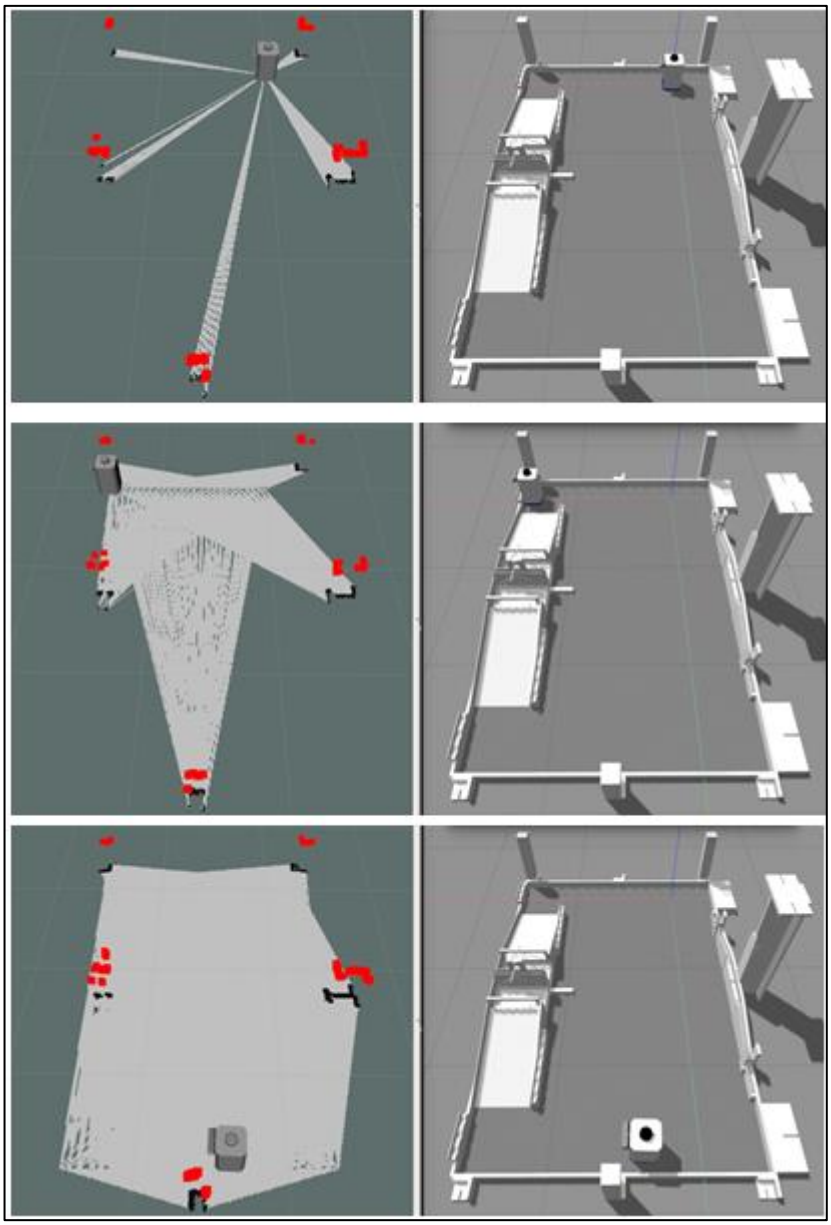

Figura 7.9 *Gmapping* amb els valors finals adoptats. (Font pròpia)

Arribats a aquest punt el canvi més evident és la freqüència de *scans* fets per unitat de distància; gràcies a altres paràmetres com *minimumScore* s'han reduït els mínims salts d'estimació que perduraven en el *package* original; amb els paràmetres *maxUrange* i *maxRange* ja des del paquet original s'evita la lectura de camp existent rere els obstacles visibles dins del rang de visió, no obstant la modificació no deixa de ser obligatòria per poder recrear els successos reals en les simulacions amb la mínima diferència possible.

7.3. Parametrització AMCL i costmaps

Com es comenta a l'inici de l'apartat anterior, es parteix d'un *package* existent i es modifica fent ús d'un altre amb millors prestacions. Inicialment es presenta la configuració original per poder visualitzar el procés de millora de tal manera poder comprendre més el funcionament d'aquests *packages* que són clau per dur a terme un càlcul de trajectòries correcte.

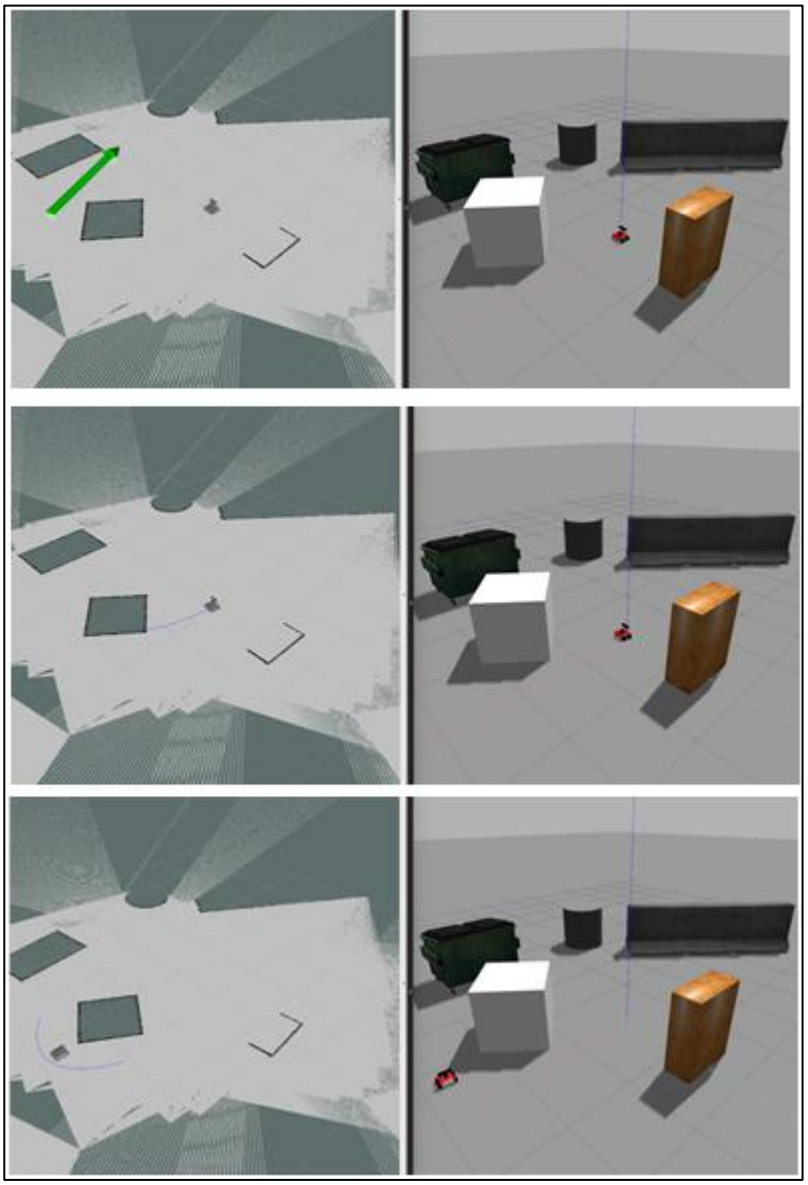

Figura 7.10 AMCL en marxa sobre el camp escanejat amb la configuració inicial de *gmapping*, la configuració utilitzada de AMCL també n'és l'original. (Font pròpia)

Per una banda el paquet de Rosbot incorporava un *gmapping* funcional en tots els aspectes, que a causa de la modificació feta per aquest projecte el resultat no difereix en excés. Per altre banda,

apareixen múltiples mancances a l'hora de fer anar al *package* AMCL el qual a més s'encarrega de fer funcionar *move_base* i el tipus de *path planning* que hi hagi configurat. Sense tenir en compte el càlcul de trajectòries, si s'observa la Figura 7.10 les imatges de l'esquerra corresponen a les de la dreta, les primeres representat la visualització des de Rviz mentre que les segones des de Gazebo; en detall mentre que les imatges de Gazebo romanen immòbils en les de Rviz es perceben canvis, desplaçaments concretament.

Des d'aquesta perspectiva és complicat veure-ho en totalitat, per això en la figura següent és possible entendre millor la problemàtica. Aquesta imatge (Figura 7.11), a diferència de l'anterior incorpora la visualització dels *costmaps*, amb els quals s'observa més clarament els *jumps* degut a la configuració de AMCL. En un principi la causa podria provenir de la mala comunicació entre el mapa prèviament escanejat i l'algoritme *Monte Carlo Localization*; és possible que si s'escaneja un mapa i després se'n fa ús d'un de diferent, a l'hora d'executar AMCL causi aquests desplaçaments. De totes maneres aquest no és el cas.

Un altre aspecte a mencionar és referent al *costmap*, com s'explica en l'apartat 6.3.3 aquest tipus de mapa tracte d'indicar el cost que representa passar vora un obstacle registrat. Es pot donar el cas d'obstacles que el LIDAR, o el sensor que s'utilitzi, només en pugui veure una part sense visualitzar la base, la qual podria ser major a la part de l'obstacle visible pel sensor. Com és el cas d'aquest projecte interessa doncs proporcionar un valor elevat a aquest cos de tal manera que el robot pugui evitar l'obstacle sense necessitat de veure'l. Aquest és un mètode poc convencional ja que és necessari tenir un espai controlat on els obstacles no siguin variables; és molt més útil utilitzar sensors de proximitat d'infraroig o d'ultrasons, preferiblement els primers ja que proporcionen menys soroll i més certesa en les lectures. No obstant, el robot original físic incorpora aquests tipus de sensors encara que de totes maneres no s'han pogut implementar en el projecte.

Per altre banda, tot i que sigui preferible requerir un alt cost pels obstacles, tenint un espai reduït com en el cas del camp d'Eurobot del qual se'n fa ús, aquest alt cos impossibilitaria moviments propers als límits del camp.

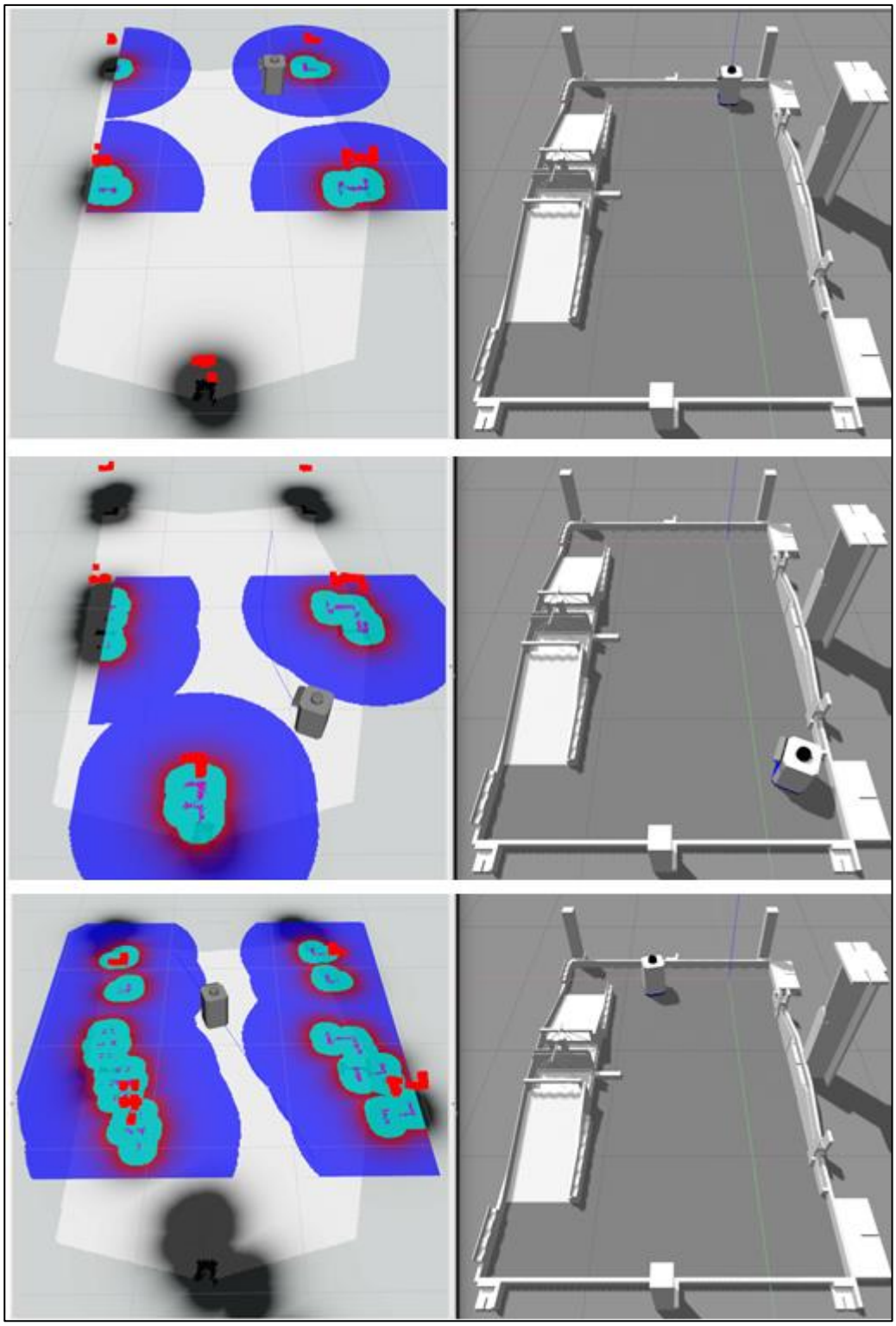

Figura 7.11 AMCL en marxa amb la configuració original, sobre el camp d'Eurobot i amb el robot nou. (Font pròpia)

Paràmetres originals	valor	Paràmetres *Turtlebot*	valor
odom_frame_id	odom	odom_frame_id	odom
odom_model_type	diff_corrected	odom_model_type	diff
base_frame_id	base_link	base_frame_id	base_footprint
update_min_d	0,5	update_min_d	0,25
update_min_a	1	update_min_a	0,2
		global_frame_id	map
		laser_max_range	12
		gui_publish_rate	10
		laser_max_beams	60
		min_particles	100
		max_particles	2000
		kld_err	0,05
		kld_z	0,99
		odom_aplha1	0,2
		odom_aplha2	0,2
		odom_aplha3	0,2
		odom_aplha4	0,2
		odom_aplha5	0,1
		laser_z_hit	0,5
		laser_z_short	0,05
		laser_z_max	0,05
		laser_z_rand	0,5
		laser_sigma_hit	0,2
		laser_lambda_short	0,1
		laser_model_type	likelihood_field
		laser_likelihood_max_dist	2
		resample_interval	1
		transform_tolerance	1
		recovery_alpha_slow	0
		recovery_alpha_fast	0
		initial_pose_x	0
		initial_pose_y	0
		initial_pose_a	0
		use_topic_map	false

Taula 7.4 Comparació entre els paràmetres originals i els de *Turtlebot* de AMCL. En vermell els valors diferents, en verd els valors i paràmetres nous, i en groc els paràmetres amb els valors per defecte. (Font pròpia)

Entre tots els paràmetres cal destacar-ne:

- *odom_frame.* Com en el cas del SLAM el valor d'aquest paràmetre ha de ser el tòpic d'odometria, és a dir han de tenir el mateix valor.
- *odom_model_type.* Un robot pot ser holonòmic o no ser-ho, per tant a partir dels quatre tipus que ROS facilita s'ha d'indicar quin d'ells es vol pel robot: *diff*, *diff-corrected*, *omni* o *omni-corrected*.
- *base_frame_id.* De la mateixa manera que amb *odom_frame*, aquest paràmetre i tots els seus homòlegs han de tenir el mateix valor indicant el nom de la base del robot, des de la qual *tf* s'encarrega de calcular les diferents distàncies respectives als altres elements tant del mateix robot com de l'espai.
- *update_min_d.* És tracte d'un valor que estableix la freqüència amb la qual es fa una càrrega de dades del filtre que s'està utilitzant; concretament el paràmetre indica en metres cada quanta distància s'aplica aquesta càrrega.
- *update_min_a.* Recorregut en graus de gir que ha de fer el robot abans de fer una càrrega de dades del filtre configurat.
- *odom_aplha1, odom_alpha2, odom_alpha3, odom_alpha4, odom_aplha5.* Són un conjunt d'estimacions de soroll en diferents parts de l'odometria del robot respecte altres parts de la mateixa. Segons el tipus d'odometria escollida aquestes estimacions prenen un valor o un altre. Precisament el cinquè paràmetre es únicament d'ús per el tipus *omni* o *omni-corrected* és a dir holonòmic; el seu valor es tradueix en una tendència en el robot per preferiblement realitzar moviments transversals i perpendiculars cap a l'objectiu, sense girar.

La resta de paràmetres corresponen a la configuració del soroll Gaussià produït pel propi sensor i les seves lectures. ROS incorpora aquest comportament per simular la realitat dels sensors fent que produeixin soroll i les lectures puguin ser afectades.

Paràmetres *Turtlebot*	valor	Paràmetres finals	valor
odom_frame_id	odom	odom_frame_id	odom
odom_model_type	diff	odom_model_type	omni_corrected
base_frame_id	base_footprint	base_frame_id	base_footprint
update_min_d	0,25	update_min_d	0,5
update_min_a	0,2	update_min_a	1
global_frame_id	map		
laser_max_range	12	laser_max_range	12
gui_publish_rate	10	gui_publish_rate	10
laser_max_beams	60	laser_max_beams	60
min_particles	100	min_particles	100
max_particles	2000	max_particles	2000
kld_err	0,05	kld_err	0,05
kld_z	0,99	kld_z	0,99
odom_aplha1	0,2	odom_aplha1	0,005
odom_aplha2	0,2	odom_aplha2	0,005
odom_aplha3	0,2	odom_aplha3	0,01
odom_aplha4	0,2	odom_aplha4	0,005
odom_aplha5	0,1	odom_aplha5	0,003
laser_z_hit	0,5	laser_z_hit	0,5
laser_z_short	0,05	laser_z_short	0,05
laser_z_max	0,05	laser_z_max	0,05
laser_z_rand	0,5	laser_z_rand	0,5
laser_sigma_hit	0,2	laser_sigma_hit	0,2
laser_lambda_short	0,1	laser_lambda_short	0,1
laser_model_type	likelihood_field	laser_model_type	likelihood_field
laser_likelihood_max_dist	2	laser_likelihood_max_dist	2
resample_interval	1	resample_interval	1
transform_tolerance	1	transform_tolerance	1
recovery_alpha_slow	0	recovery_alpha_slow	0
recovery_alpha_fast	0	recovery_alpha_fast	0
initial_pose_x	0	initial_pose_x	0
initial_pose_y	0	initial_pose_y	0
initial_pose_a	0	initial_pose_a	0
use_topic_map	false	use_topic_map	false

Taula 7.5 Comparació entre els paràmetres de *Turtlebot* i els finalment adoptats. En vermell els valors diferents i en verd els valors nous. (Font pròpia)

En definitiva aquest és el resultat final de la configuració de AMCL, ara bé aquest *package* treballa conjuntament amb els *costmaps*, per tant abans de donar aquest apartat per acabat cal configurar els arxius corresponents amb la finalitat de poder visualitzar un mapa amb *costmaps* correctament definit.

Concretament *costmap* es pot dividir essencialment en tres capes, tot i que puguin existir-ne més. Aquestes capes són:

- *Static Layer*. S'encarrega de controlar i parametritzar aquelles dades externes immutables, per exemple el tòpic */map*.
- *Obstacle Layer*. Es subscriu al tòpic del sensor per poder controlar els obstacles, referint-se a la seva posició, dibuixar-los en el mapa o esborrar-los d'aquest. Pot funcionar amb un *plugin* que treballa en dues dimensions (*ObstacleCostmapPlugin*) o amb un de tres dimensions (*VoxelCostmapPlugin*).
- *Inflation Layer.* És l'encarregada de donar un valor al voltant dels obstacles i és el que es visualitza en definitiva.

Aquestes capes podrien definir-se totes en un mateix arxiu d'extensió YAML però en la gran majoria dels casos és més útil separar certa informació en dos grups, un grup anomenat *global costmap* i un altre anomenat *local costmap*. D'aquesta manera es pot decidir mostrar un informació d'una manera o altre o no mostrar-la.

Tot i separar-se en dos arxius, aquests normalment incorporen una quantitat mínima del total mentre que un tercer arxiu (*costmap common params*) defineix la resta de paràmetres que serveixen per ambdós grups. És per tant aquest el més important i des del qual s'han fet les modificacions.

D'aquest arxiu és essencial destacar alguns paràmetres com

- *footprint*. Aquest defineix la forma de la base del robot. *Costmap* per tenir constància de com és el robot, per poder treballar millor les distàncies entre robot i obstacle, no fa servir el model 3D del mateix robot; en canvi utilitza una forma 2D la qual defineix la base del robot (*footprint*).

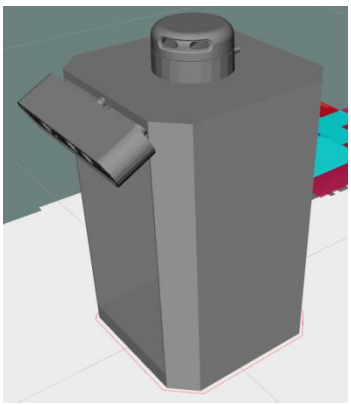

Figura 7.12 En la base del robot el *footprint* en color vermell. (Font pròpia)

- *map_type.* Hi ha de dos tipus, el que treballa en dues dimensions (*Obstacle*) i el que ho fa en tres dimensions (*Voxel*). Des dels paràmetres de *Turtlebot* el tipus de mapa escollit és el *Voxel* per tant s'ha deixat d'aquesta manera doncs no afecta a les condicions del robot i aparentment no hi ha diferència en la visualització.
- *max_obstacle_height.* En metres la màxima distància que un obstacle pot fer d'alt per inserir-lo en el mapa, aquest paràmetre ha de ser poc més elevat que l'altura del robot. Sent el robot de 0,3 m i el LIDAR de 0,05 m s'estableix aquest valor a 0,4 m.
- *obstacle_range.* Distància màxima en metres a la que es pot ubicar un obstacle, sent el mapa de 3 m per 2 m s'estableix al valor màxim és a dir 3 m.
- *raytrace_range.* Distància màxima en metres a la que es pot esborrar un obstacle, sent el mapa de 3 m per 2 m s'estableix al valor màxim és a dir 3 m.
- *cost_scaling_factor.* És la variable d'una equació que estableix el valor del cost d'un obstacle depenent de la distància que es trobi aquest del robot.

$$exp(-1.0 * cost\_scaling\_factor * (distance\_from\_obstacle - inscribed\_radius)) * (costmap\_2d::INSCRIBED\_INFLATED\_OBSTACLE - 1) \quad \textbf{(Eq. 7.2)}$$

- *inflation_radius.* Radi en metres de la inflació dels obstacles.

A partir dels últims dos paràmetres s'han fet un seguit de proves amb les quals s'ha acabat definint aquest apartat:

	config1	config2	config3	config4	config5
cost_scaling_factor	5	2	8	5	5
inflation_radius (m)	0,15	0,15	0,15	0,3	0,1

Taula 7.6 Proves realitzades per trobar la configuració òptima del *costmap*. (Font pròpia)

Per la naturalesa de l'equació anterior ((Eq. 7.2) augmentant el primer dels paràmetres (*cost_scaling_factor*) s'obtenen valors cada vegada més petits de tal manera que per menys distància entre obstacle i robot hi haurà un valor x; mentre que si es disminueix s'obtenen cada vegada valors més grans així que per menys distància entre obstacle i robot hi haurà un valor major a x. En definitiva si es vol mantenir el robot lo més lluny possible dels obstacles, aquest valor (*cost_scaling_factor*) ha de ser com més petit millor.

En *Turtlebot* aquests paràmetres prenen el valor de 5 i 0,5 m respectivament, tot i així sabent que els robots poden tenir fins a 300 mm de base (perímetre màxim de 1200 mm entre 4 costats), es decideix partir de 0,15 m de radi.

Per portar a terme aquesta prova s'han sotmès tots els casos de configuració sota les mateixes condicions. Es fa partir el robot del punt d'origen, fent-lo passar pel costat d'un robot enemic que ha acabat de aparèixer i se'l fa arribar a l'altre banda camp.

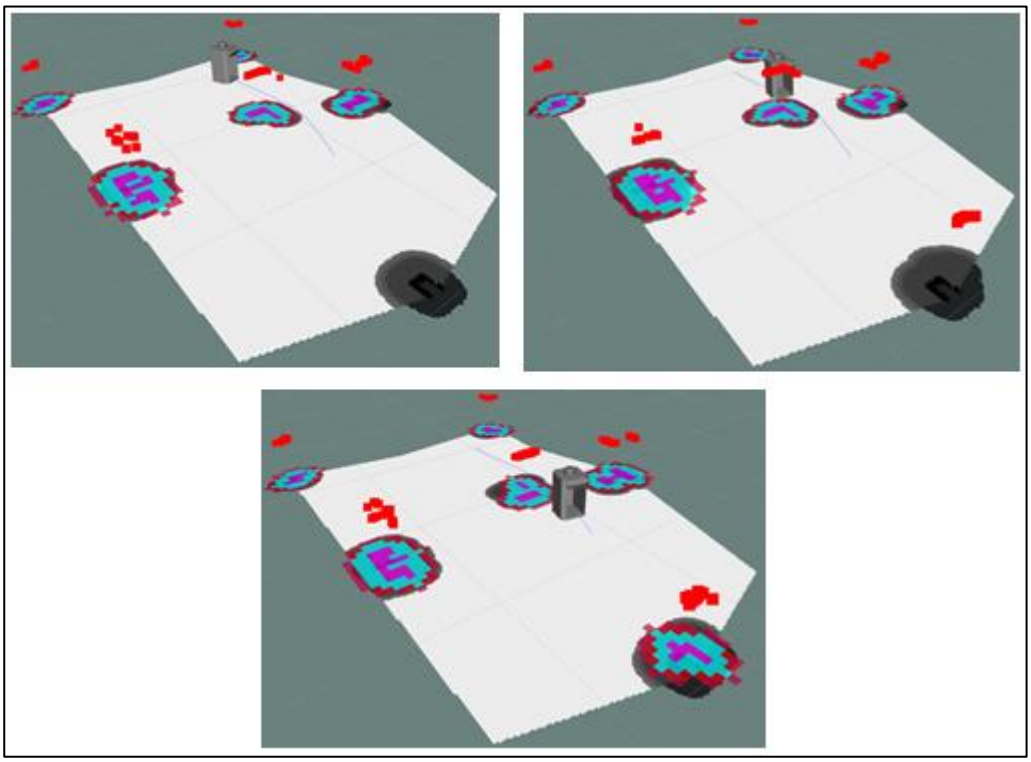

Figura 7.13 Primera configuració. Cost_scalating_factor = 5; inflation_radius = 0,15. (Font pròpia)

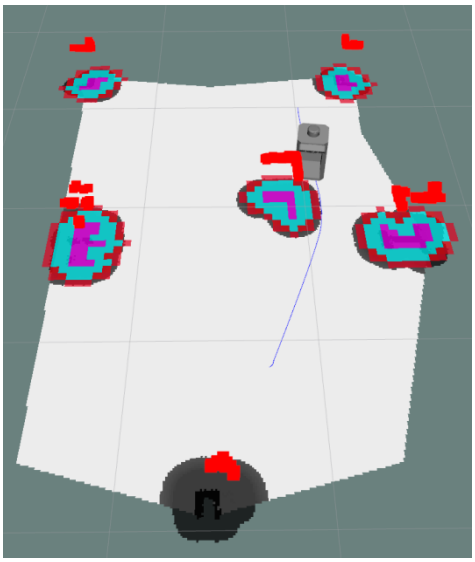

Figura 7.14 Segona configuració. Cost_scalating_factor = 2; inflation_radius = 0,15. (Font pròpia)

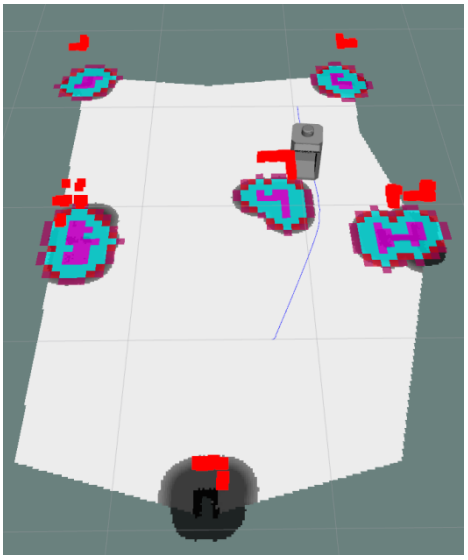

Figura 7.15 Tercera configuració. Cost_scalating_factor = 8; inflation_radius = 0,15. (Font pròpia)

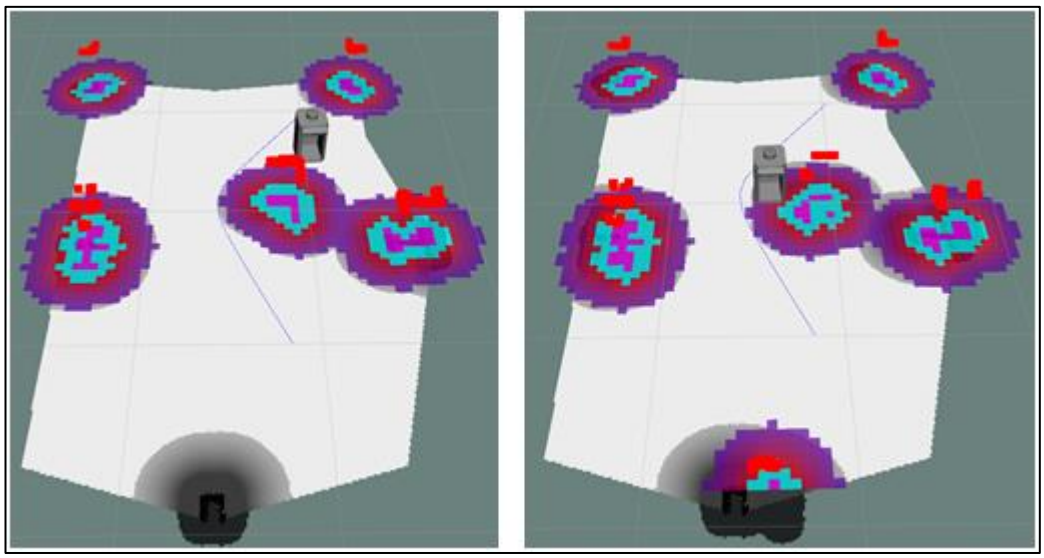

Figura 7.16 Quarta configuració. Cost_scalating_factor = 5; inflation_radius = 0,3. (Font pròpia)

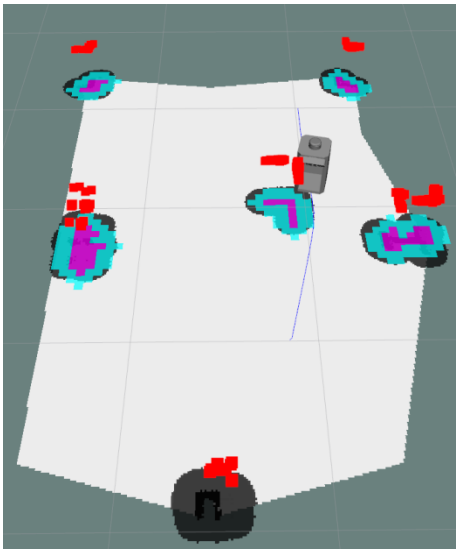

Figura 7.17 Cinquena configuració. Cost_scalating_factor = 5; inflation_radius = 0,1. (Font pròpia)

Un cop fetes les proves, de les cinc, és fàcil descartar-ne dues; En la Figura 7.16 es mostra la quarta configuració on s'ha mantingut el valor del cost d'escalat però s'ha augmentat el valor del radi d'inflació, es evident com es pot observar que amb aquesta grandària d'inflació és molt complicat que el *path planning* funcioni correctament. Degut a aquest augment del radi, el camí resultant ha canviat respecte a les altres proves cosa que és un inconvenient ja que resulta que no era el camí més curt real.

L'altre configuració a descartar és l'última la qual s'ha fet a la inversa reduint el valor del radi. En aquest cas el radi es tan petit que el robot intenta passar ben a prop i xoca contra l'altre robot, lo que en cas de ser un cas real suposaria una sanció en la competició i pèrdua dels punts en la partida. El xoc es produiria en el cas de que el robot no disposes de mesures per evitar-ho, com s'ha comentat abans sensors de proximitat.

Per altre banda de les altres configuracions se'n extreu que no s'hi nota cap diferència, tan una com altre de les tres varien en el valor del cost d'escalat lo que suposaria un augment del cost quan és 2 i una disminució a l'hora de ser 8. Vistos els resultats la configuració final resulta ser la primera, donat que el primer paràmetre no dona cap resultat visual aparentment es segueix utilitzant el valor inicial doncs no és ni massa alt ni massa baix, per altre banda el radi d'inflació ja s'ajustava des d'un principi per tant es manté.

7.4. Parametrització *move_base* i *path planning*

Un dels altres aspectes del *package* original era la nul·la configuració de *move_base* el qual per sobre de tot procura que el robot no es quedi encallat. Aquest *package* a més s'encarrega d'escollir quins són els *path planners* actius en el moment, des de l'arxiu de configuració d'extensió YAML es pot personalitzar el mode en que el robot fa el seu camí (*path planner*) i el mode en que es recupera d'estar encallat.

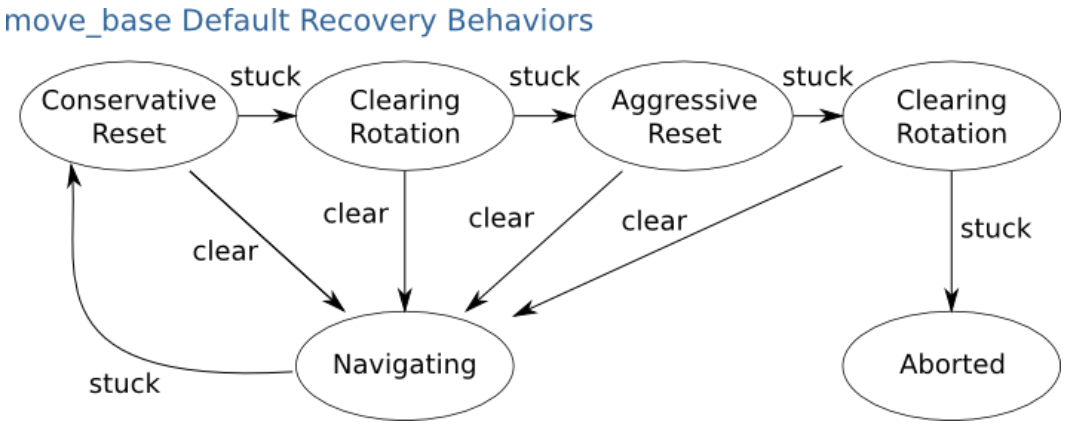

Figura 7.18 Configuració per defecte de *move_base* respecte al *recovery behavior*. (Font: (Open Source Robotic Fundation, 2011))

Quan el robot topa amb un obstacle pot quedar encallat intentant sortir endavant però evidentment no és possible. Així doncs s'incorpora un mode en el que es segueix una pauta en el cas que el robot no pugui avançar, aquesta pauta és la que es mostra en la figura anterior: primerament es neteja el mapa d'obstacles i es torna a refer amb una rotació en el lloc tornant després a intentar recuperar el camí, si no funciona es repeteix el procés fent una neteja d'obstacles més agressiva. Finalment en cas que el robot torni a detectar-se com a encallat es declarà que l'objectiu és inabastable.

Un cop configurat el *recovery_behavior* amb els paràmetres per defecte de ROS, ja que *Turtlebot* no incorpora uns paràmetres propis, es comença un seguit de proves primer per comprovar quin *base_global_planner* és millor pel projecte.

	config1	config2	config3
base_global_planner	Global_Planner	Navfn	Navfn
base_local_planner	DWA	DWA	Eband

Taula 7.7 Taula de configuracions fetes per testejar primer el millor *base_global_planner* i després el millor *base_local_planner*. (Font pròpia)

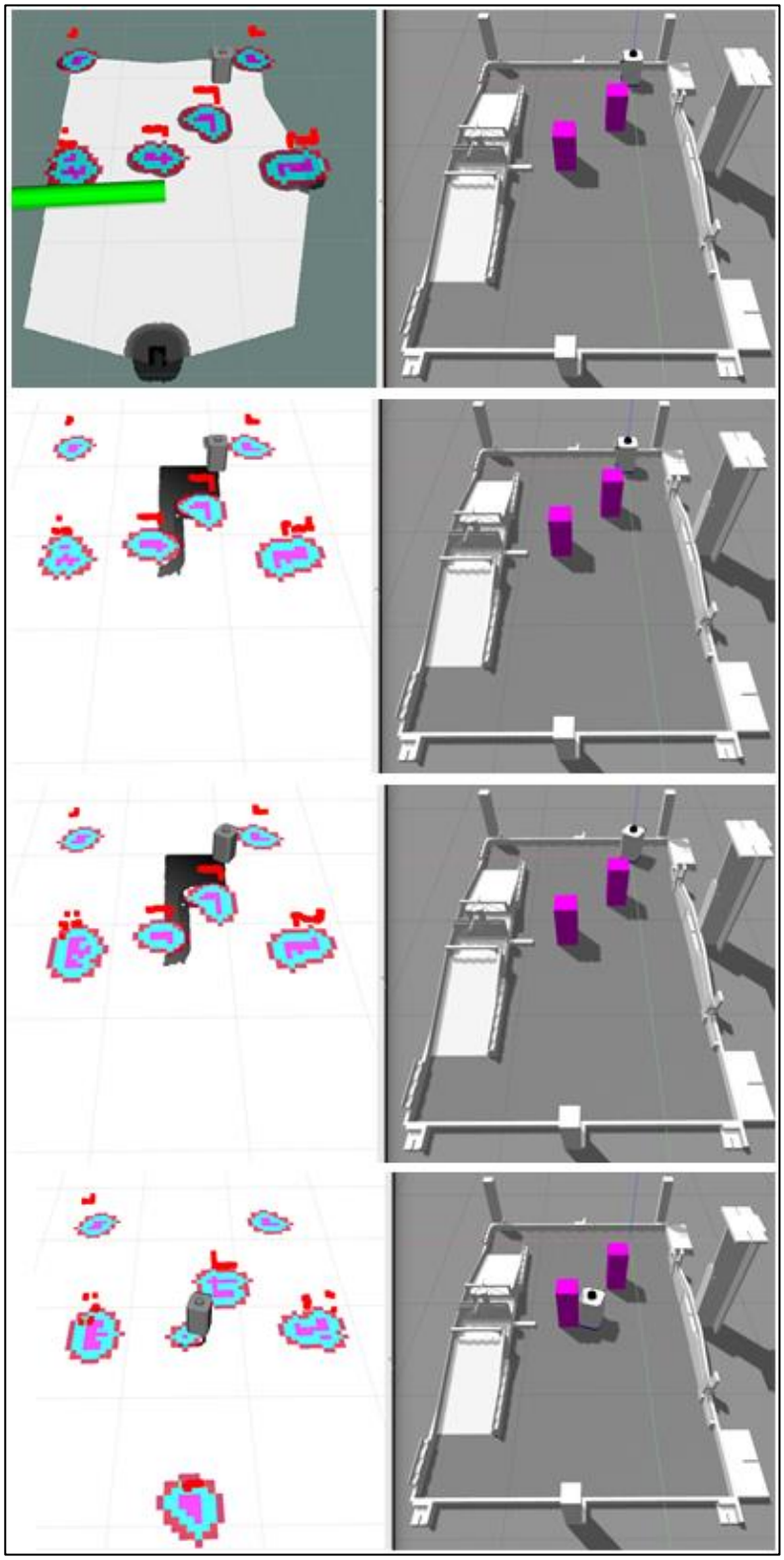

Figura 7.19 Primera configuració. *Base_global_planner* = *Global_planner*; *base_local_planner* = *dwa_local_planner*. (Font pròpia)

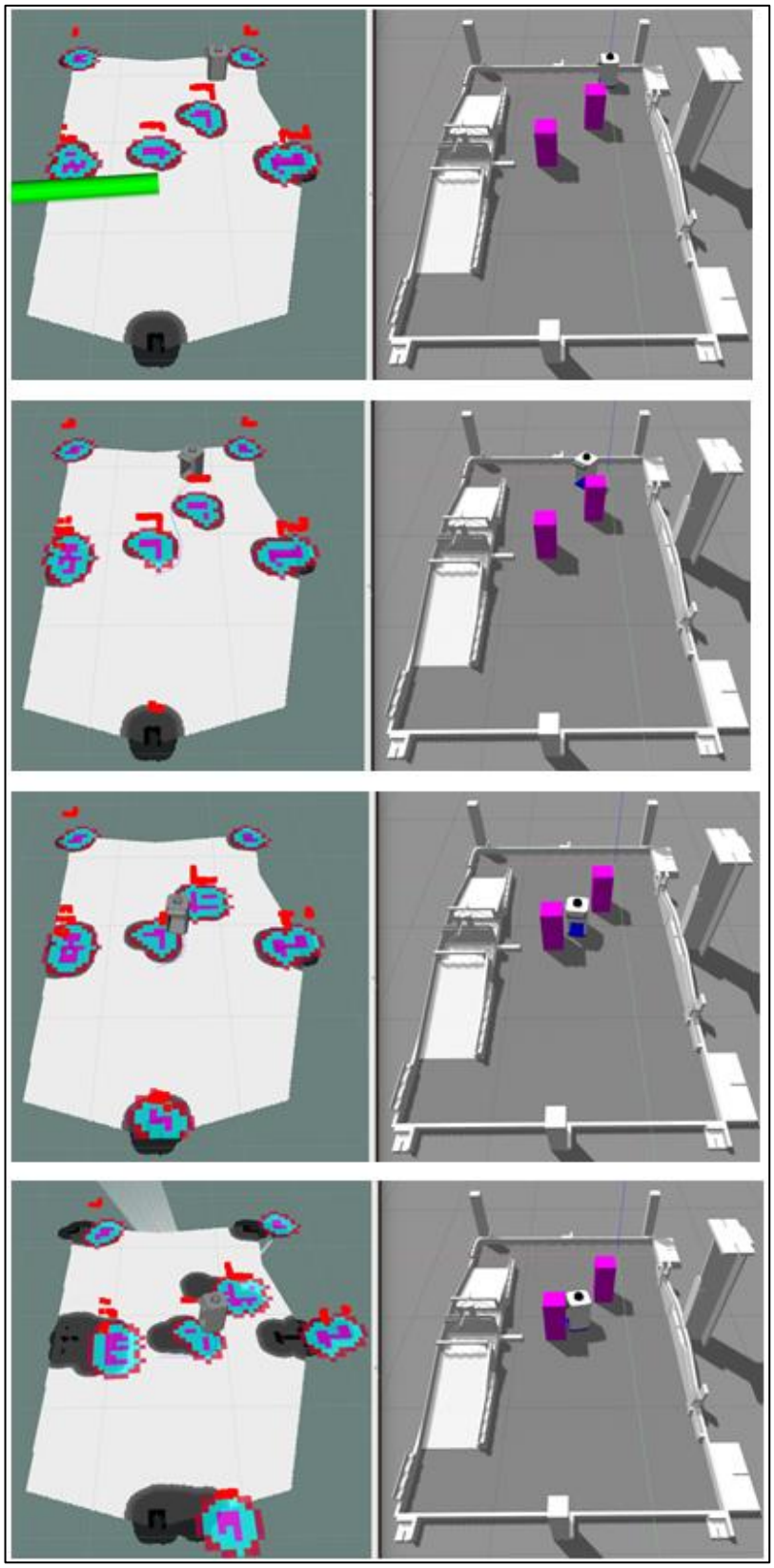

Figura 7.20 Segona configuració. *Base_global_planner* = Navfn; *base_local_planner* = *dwa_local_planner*. (Font pròpia)

Per portar a terme aquesta comparació s'ha exposat al robot a les mateixes condicions amb els dos tipus d'algoritmes. Essent la prova una de considerada la més problemàtica; en qualsevol cas si el robot és capaç de superar correctament una prova no resulta en cap resultat concloent doncs poden haver-hi moltes casuístiques, en canvi si s'exposa al robot en un ambient difícil el fet de superar la proba pot significar que una altre de menys dificultat la pugui superar sense entrebancs. Així doncs, com es mostra en les figures anteriors: es fa començar al robot en el punt d'origen; se li ordena que es mogui fins a la posició indicada per l'extrem de la fletxa verda que s'observa en les primeres imatges de les Figures Figura 7.19 i Figura 7.20; i seguidament el robot fa ús del seu *path planning* configurat i traça una ruta. El motiu d'estudi en aquestes proves són els camins traçats no precisament el fet que el robot arribi o no a l'objectiu tot i que també és un punt que es pot tenir en compte.

Inicialment es volia plantejar un altre model de proves en el que s'incorporessin variacions en la configuració de *Global Planner*, no en Navfn ja que no té paràmetres de configuració. Un inconvenient en la instal·lació o degut a la versió que s'utilitza s'ha trobat una complicació respecte al primer algoritme. Si s'observa detingudament la Figura 7.19, en la tercera imatge el robot està d'esquenes al camí traçat i es que ha entrat en mode *recovery_behavior*, no se li ha trobat una explicació ha aquest succés no obstant no és aïllat, en més d'una ocasió amb diferents configuracions d'aquest algoritme i amb proves molt més senzilles el resultat és gairebé el mateix. A l'hora d'indicar el *goal* el robot triga cert temps en calcular el camí òptim, de totes maneres en la figura mencionada s'observa amb el paràmetre *visualize_potential* (apartat 6.3.3) el camí calculat i no és fins un segons més tard que el robot comença a moure's, després de sortir del *recovery_behavior* si és dona el cas. En l'última imatge es troba que el robat s'ha tornat a encallar i retorna al *recovery_behavior*, en aquest punt es decideix després de molts intents deixar de banda aquest algoritme i treballar únicament amb Navfn.

No obstant no s'ha deixat de fer proves amb l'algoritme que integra el de Dijkstra. En la Figura 7.20 el resultat que dóna l'algoritme coincideix amb el calculat pel *Global_planner*, en detriment d'aquest tot i la problemàtica, el temps d'execució de Navfn és molt mínim per tant hauria de tractar-se d'un càlcul extrem on es notaria la diferència. Tot i que no s'arribi a la destinació el càlcul era el correcte per tant es dona per bo.

Pel que fa al *base_local_planner* les proves que s'han fet són poques ja que són pocs els algoritmes que siguin capaços d'acceptar un robot holonòmic. D'entre els predeterminats de ROS s'ha trobat un que sí ho integra, es tracte de *eband_local_planner*.

Aquest *package* utilitza el mètode de banda elàstica que segons els seus autors:

"Proposem un nou *framework* per tapar el forat entre *path planning* i *control*. La idea es implementar moviments a partir de sensors locals deformant en temps real el camí generat pel *planner*. Nosaltres anomenem aquesta deformació lliure de col·lisions una banda elàstica" (Quinlan i Khatib, 1993)

Referint-se a *path plannig* com l'encarregat de calcular la ruta òptima i amb *control* referint-se a la regulació existent entre les dades d'un sensor i la interacció en el món real degut a aquestes dades. D'aquesta manera aquest mètode actua en temps real sense necessitat de fer cridar el *path planner* per re-calcular la ruta en cas que es necessiti. En tot cas això seria de molta utilitat si es tractés d'un *base_global_planner*, no obstant segons la seva descripció pren la naturalesa d'un *base_local_planner* que actua amb un camí establert i sense poder canviar aquest.

Per configurar aquest algoritme només ha bastat amb la variació d'un paràmetre, *costmap_weight*, és un factor que es fa servir per calcular la distància als obstacles. En altres paraules, a partir del seu valor el robot es desplaça certa distància del camí traçat pel *path planning* quan passa vora un obstacle.

En aquest cas s'han fet dos tipus de proves, la segona és la mateixa utilitzada per comparar els *base_global_planners*, aquest cop tenint molt en compte si el robot es xoca i si arriba al punt final o no. La primera en canvi només s'utilitza un obstacle per estudiar la distància efectiva entre robot i obstacle pel fet de variar el paràmetre.

	config1	config2	config3	config4	config5
costmap_weight	5	10	15	12,5	20
xocs 1 obstacle	1	0	0	0	1
xocs 2 obstacles	-	2	1	2	0
arriba a destí	O-	OX	OO	OX	OO
distància a obstacle	xoc	molt a prop	lluny	suficient	molt lluny

Taula 7.8 Comparativa de configuracions segons les diferents proves fetes. (Font pròpia)

De la taula anterior:

- O. Significa que ha arribat a destí, si apareix en primera posició es refereix a la primera prova, si en canvi és en segona posició es refereix a la segona.
- X. Significa que no ha arribat a destí, segueix el mateix criteri que el símbol anterior.
- -. Significa que no ha realitzat una prova, segueix el mateix criteri que els dos símbols anteriors.

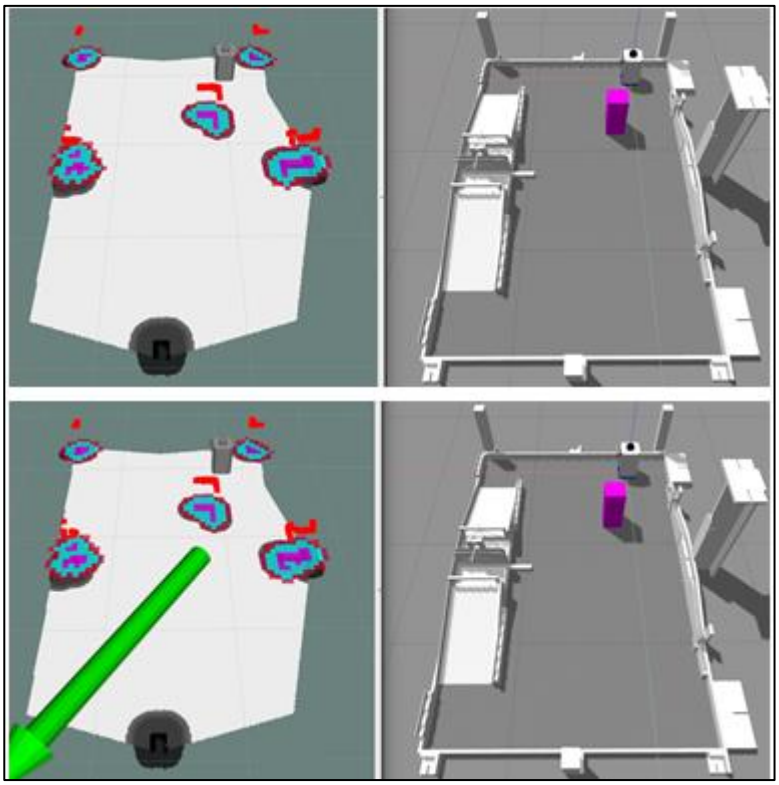

Figura 7.21 Inici de la primera prova per les diferents configuracions del *eband_local_planner*. (Font pròpia)

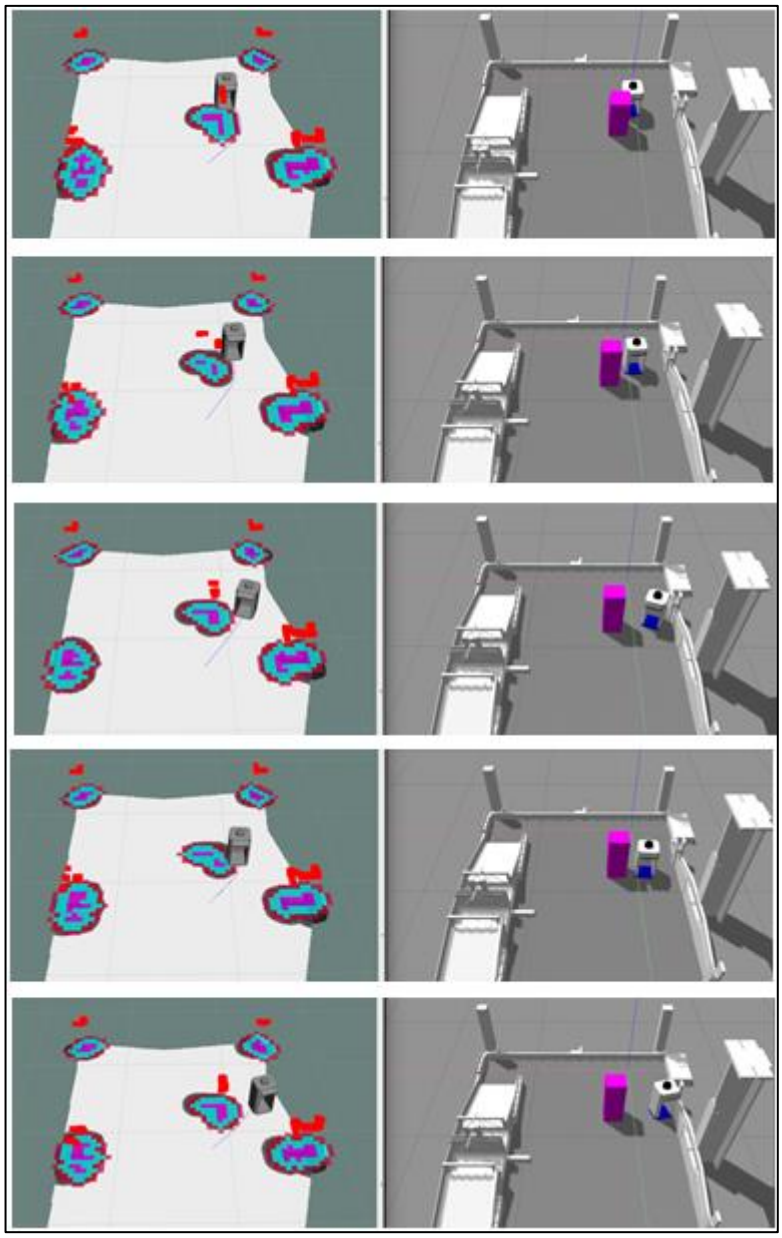

Figura 7.22 Diferents resultats per les diferents configuracions de la primera prova. (Font pròpia)

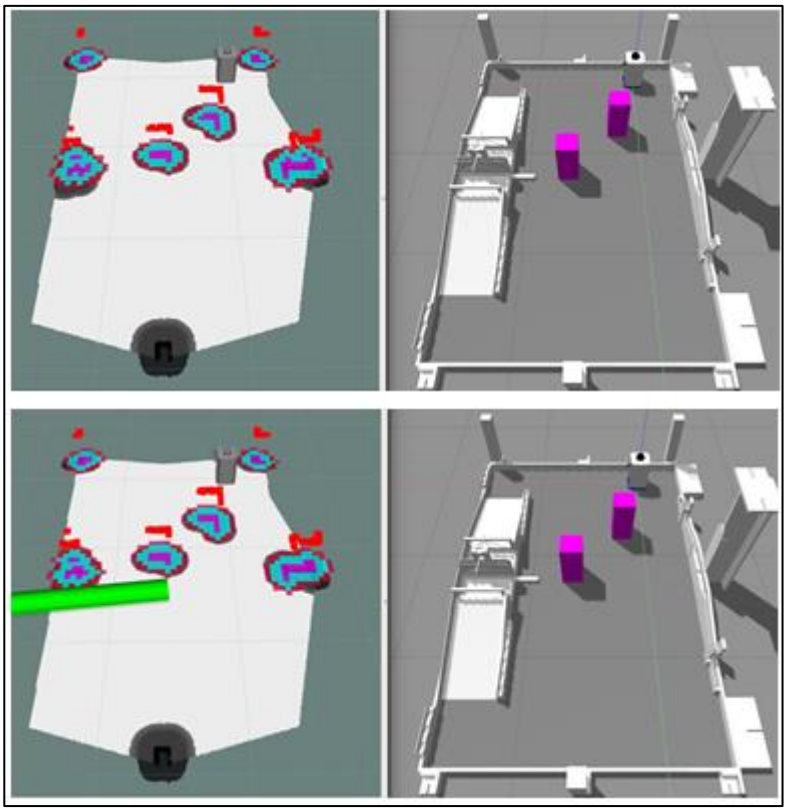

Figura 7.23 Inici de la segona prova per les diferents configuracions de *eband_local_planner*. (Font pròpia)

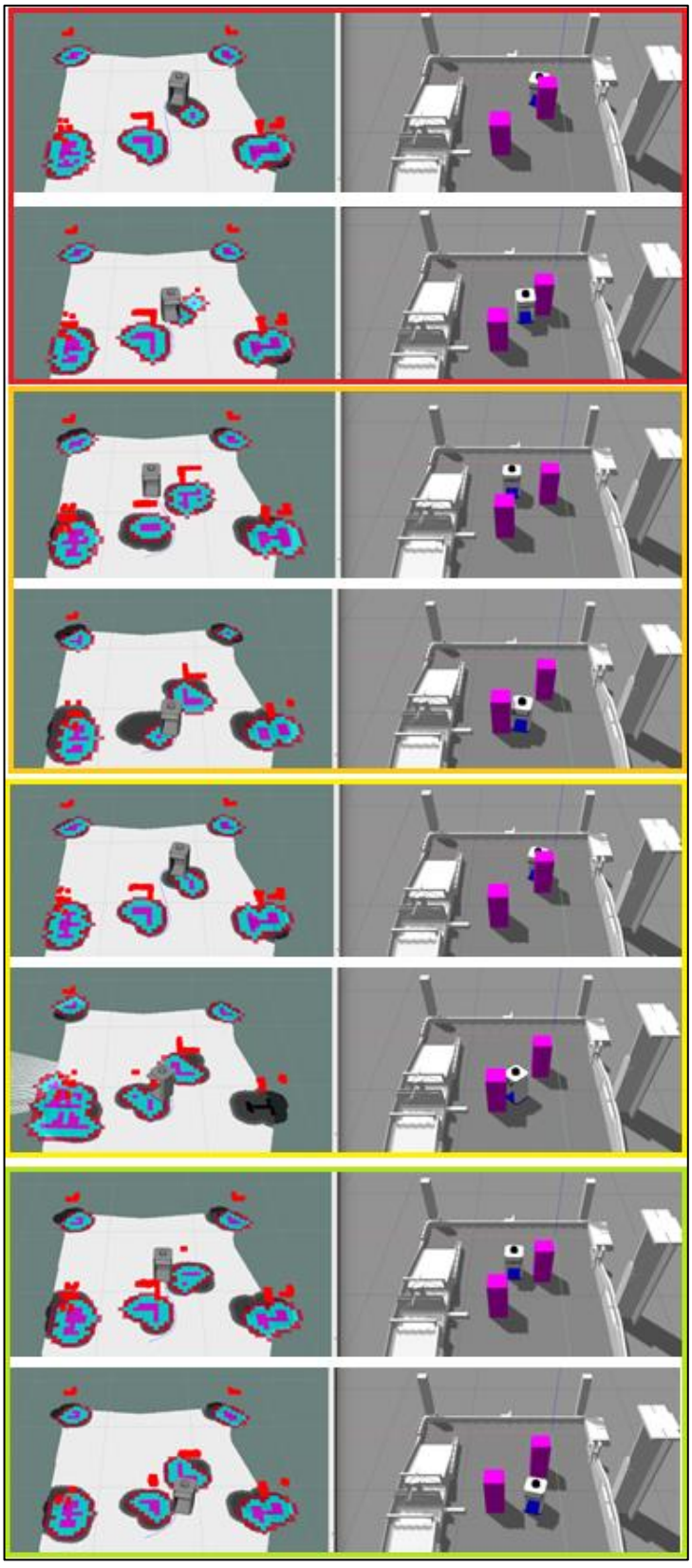

Figura 7.24 Diferents resultats per les diferents configuracions de la segona prova. Els resultats s'agrupen en imatges de dos separades per colors. (Font pròpia)

La primera prova ha valgut per comprovar el valor de *cost_weight* correcte en un principi, observant la Figura 7.22 es pot eliminar per complet la configuració inicial de valor 5 doncs ha ocorregut un xoc amb el suposat robot contrincant, lo que suposa una pèrdua de punts. En la resta de resultats cal destacar-ne el de valor 20 i el de 12,5. L'últim a diferència del de valor 10 té una distància efectiva suficientment correcte per evitar el robot i no fer gaire més gran el camí ja calculat, en canvi amb valor 10 tot i que no hi ha xoc és molt possible que n'hi hagi en alguna altre ocasió doncs la distància és mínima i no dona seguretat. Pel que fa a la configuració de valor 20 existeix un xoc però no és amb el robot enemic sinó amb l'estructura, la distància és massa gran doncs triga massa en passar vora l'obstacle fent el camí més llarg del que en realitat és. Pel que fa al valor de 15 és un cas entremig de 12,5 i 20, és prou lluny de l'obstacle per evitar-lo però no gaire a prop del camí a seguir per no allargar-lo.

En la segona prova el resultat ha valgut per acabar de discriminar i poder avaluar amb menys dades. En primera instància aquesta prova no s'ha realitzat per la primera configuració doncs havent registrat un xoc amb un senzill obstacle de la primera prova no cal exposar-lo a una de més dificultat. El valor preferit anteriorment era el de 12,5 que un cop passada la segona prova és el que ha registrat més xocs juntament amb el valor de 10 per tant es considera que no són aptes. Els valors restants són 15 i 20: amb un valor de 15 el robot ha xocat amb el segon obstacle però ha arribat a destí a diferència dels dos primers; mentre que amb un valor de 20 no hi hagut cap xoc i el robot ha arribat a destí. Per tant es considera que 20 és el valor correcte, en contrapartida la configuració de *cost_weight* establerta en 20 pot ocasionar en la majoria dels casos que el robot es desviï massa del camí calculat i en definitiva quedar-se encallat amb l'estructura del camp. Tot i així, tractant-se d'una competició premia el fet de fer el màxim de punts i no perdre'n cap, així doncs aquest és el valor buscat.

La resta de paràmetres de *eband_local_planner* són referents a les velocitats del robot, als aspectes de les bombolles de l'algoritme i a les toleràncies que marquen l'error acceptat en la posició d'arribada del robot al *goal*. Un cop s'ha marcat un destí aquest té unes coordenades de posició a les quals el robot pretén arribar però segons l'error que s'accepti el robot arribarà el destí amb més precisió o menys. Aquest paràmetre s'ajusta en el següent apartat.

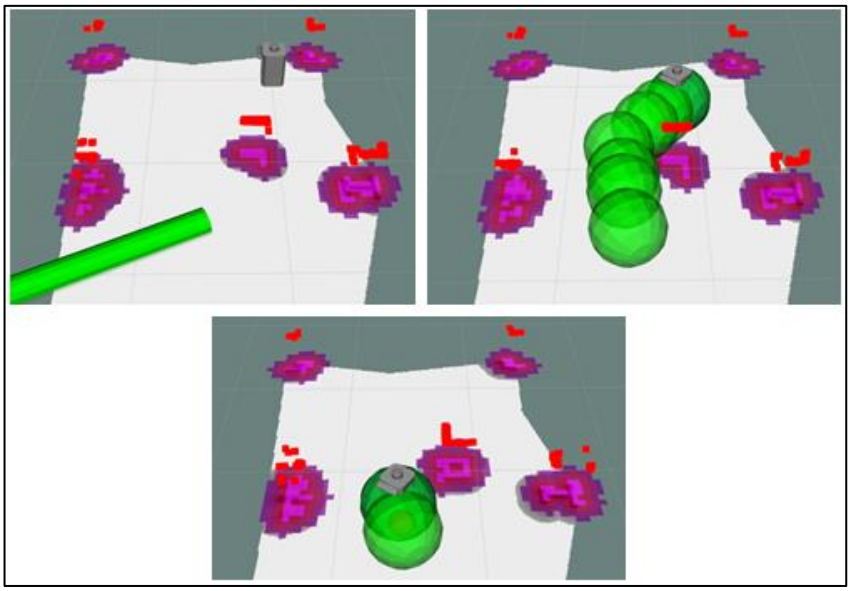

Figura 7.25 Les bombolles que caracteritzen a *eband_local_planner*. (Font pròpia)

7.5. Configuració dels punts coordinats per a *StrategySimulator*

Un cop el codi python es considerà funcional era qüestió de calibrar els punts. Per fer-ho es porta a terme un històric de proves en el que es registra des de l'inici: les coordenades de cada punt, la correcció que s'ha fet tenint en compte certs criteris, i l'avaluació del resultat. Tot i ser una simulació es vol considerar la major exactitud possible, degut a que la seva funció i finalitat pot passar a quelcom més útil, treballant de la mateixa manera però coordinadament amb el robot real per exemple d'aquesta manera és necessari que els punts siguin precisos.

	Original		Correcció 1		Correcció2		Correcció 3	
PUNTS	AVALUACIÓ	CORRECCIÓ	AVALUACIÓ	CORRECCIÓ	AVALUACIÓ	CORRECCIÓ	AVALUACIÓ	CORRECCIÓ
P1								
P2								
P3								
P4								
P5								
P6								
P7		+x						
P8		+x		-x				
P9		+x		-x		afegir step		
P10								
P11								
P12		+x		+y, -x		-y		ajust paràmetre d'aproximació
PA1		-x		-x		-x		
PE1								
PE2								

Taula 7.9 Històric de les avaluacions fetes per a cada punt registrat en l'aplicació. En verd els punts considerats correctes i en vermell els punts considerats mal calibrats, s'afegeix en cada prova un breu comentari de la correcció que s'ha de portar a terme. (Font pròpia)

En els comentaris de la taula anterior cal remarcar que per exemple "+x" significa augmentar la coordenada x mentre que "-y" per exemple significa disminuir la coordenada y.

	Original		Correcció 1		Correcció 2		Correcció 3	
PUNTS	X	Y	X	Y	X	Y	X	Y
P1	0,200	0,250	0,200	0,250	0,200	0,250	0,200	0,250
P2	0,200	1,700	0,200	1,700	0,200	1,700	0,200	1,700
P3	0,250	2,500	0,250	2,500	0,250	2,500	0,250	2,500
P4	0,450	0,500	0,450	0,500	0,450	0,500	0,450	0,500
P5	0,750	0,500	0,750	0,500	0,750	0,500	0,750	0,500
P6	1,050	0,500	1,050	0,500	1,050	0,500	1,050	0,500
P7	1,800	0,225	1,900	0,225	1,900	0,225	1,900	0,225
P8	1,343	1,000	1,447	1,000	1,395	1,000	1,395	1,000
P9	1,343	2,000	1,447	2,000	1,395	2,000	1,395	2,000
P10	0,800	1,000	0,800	1,000	0,800	1,000	0,800	1,000
P11	0,800	2,000	0,800	2,000	0,800	2,000	0,800	2,000
P12	1,400	1,300	1,500	1,300	1,450	1,360	1,450	1,320
PA1	1,500	0,225	1,450	0,225	1,400	0,225	1,350	0,225
PE1	1,200	1,300	1,200	1,300	1,200	1,300	1,200	1,300
PE2	1,200	1,720	1,200	1,720	1,200	1,720	1,200	1,720

Taula 7.10 Històric de les correccions portades a terme per a cada prova, indicant en metres els valors de les coordenades de cada punt. S'hi remarca quines són considerades correctes en verd i quines incorrectes en vermell. (Font pròpia)

Amb els valors definits en primera instància s'ha trobat que la majoria complien de manera acceptable, és a dir que no es passaven de llarg o no es quedaven curts de l'objectiu en el camp. En canvi d'altres s'ha considerat que no eren prou acceptables així que s'ha procedit a marcar-los, aquests punts són: P7, P8, P9, P12 i PA1, sent PA1 el Punt d'Accés a la zona de perill de l'equip groc. En un parell de casos en particular, que tot seguit es mostrarà, el robot xocava amb l'estructura del camp, com que l'aplicació tracte d'evitar aquest problema es tracte d'un error greu.

Val a dir que les consideracions que a partir d'aquí s'exposaran poden solucionar-se per una altre via que no sigui el calibratge dels punts coordinats. Com s'explica en l'apartat de parametrització de *move_base* i *path planning* (apartat 7.4) existeix un paràmetre que ajusta l'aproximació del robot a l'objectiu, de tal manera que segons com es modifiqui el valor, el robot pot considerar que ha arribat al seu destí a certa distància real d'aquest. Així doncs mai el robot arribarà de la mateixa manera als punts als quals s'envia al robot, en tot cas aquest aspecte ja s'ha tingut en compte i per tant els punts considerats com a incorrectes ho són per que l'error d'aproximació a l'objectiu és massa gran.

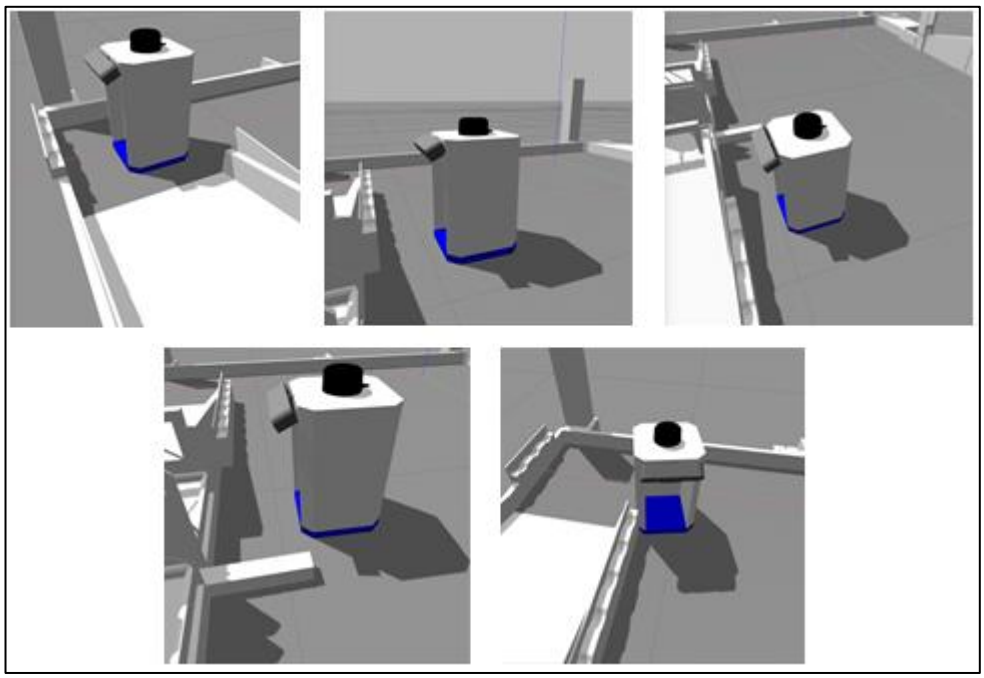

Figura 7.26 De d'alt a baix i d'esquerra a dreta les posicions, en la simulació de Gazebo, dels punts P7, P8, P9, P12 i PA1. (Font pròpia)

Es pot observar en la figura anterior la raó per la qual no s'han considerat els punts com a correctes, la gran majoria queda lluny de l'objectiu i en el cas de PA1, de tornada de P7 cap a P8, el robot xoca amb l'estructura del camp. En una situació real el robot no estaria prou a prop per manipular els discs amb les seves eines pel que fa als altres punts incorrectes.

Per calcular les coordenades dels punts es fa servir el mapa de la figura següent on el punt d'origen esta, com es pot observar, en la cantonada inferior esquerra. Per tots els camps "mapejats" el punt d'origen és un de diferent, per aquest motiu en l'aplicació ja es té en compte aquest aspecte, es treballa doncs amb l'inici com mostra la Figura 7.27 afegint un offset que ve a ser la diferència de x i de y per cada punt inicial.

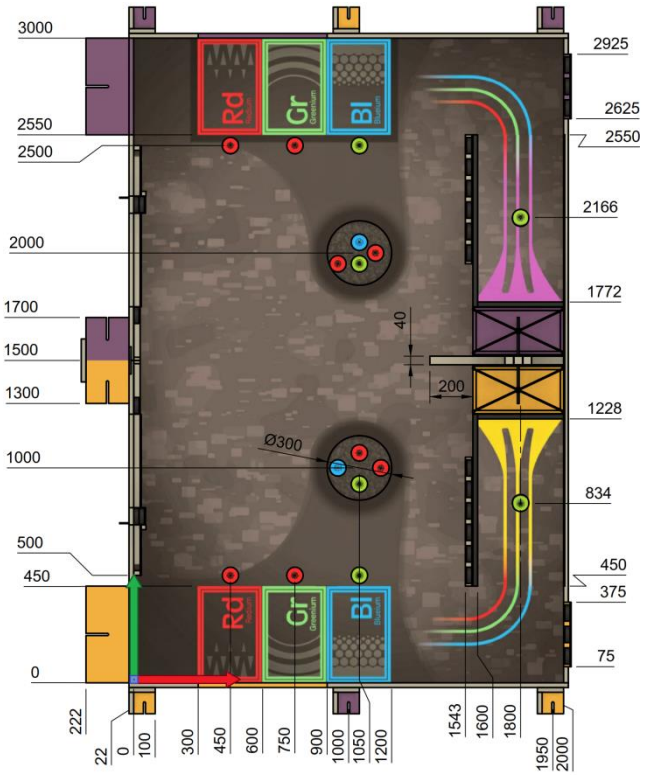

Figura 7.27 Camp d'Eurobot amb les seves mesures, en la cantonada inferior esquerra del camp s'indica el punt d'origen des d'on es prenen els valors dels punts coordinats, l'eix verd és correspon a l'eix d'ordenades i l'eix vermell al d'abscisses. (Font: (Eurobot, 2019b))

Un cop realitzada la primera prova es considera que els punts estan massa allunyats com ja s'ha comentat, per aquesta raó s'ha portat a terme la primera correcció en la que s'argumenta que el robot ha d'estar com a mínim a una distància de l'objectiu igual o major a la meitat de la base del robot. És a dir, si el robot té una base de 190,5 mm la distància entre, per exemple, el distribuïdor de discs del punt 8 (P8) i el robot ha de ser com a mínim de 190,5/2 per tant 95,25 mm. Així doncs s'ha aplicat una correcció de 95,25 mm per P8 i P9 i de 100 mm per P7 i P12. En el cas de PA1 s'ha aplicat una correcció objectiva de -50 mm.

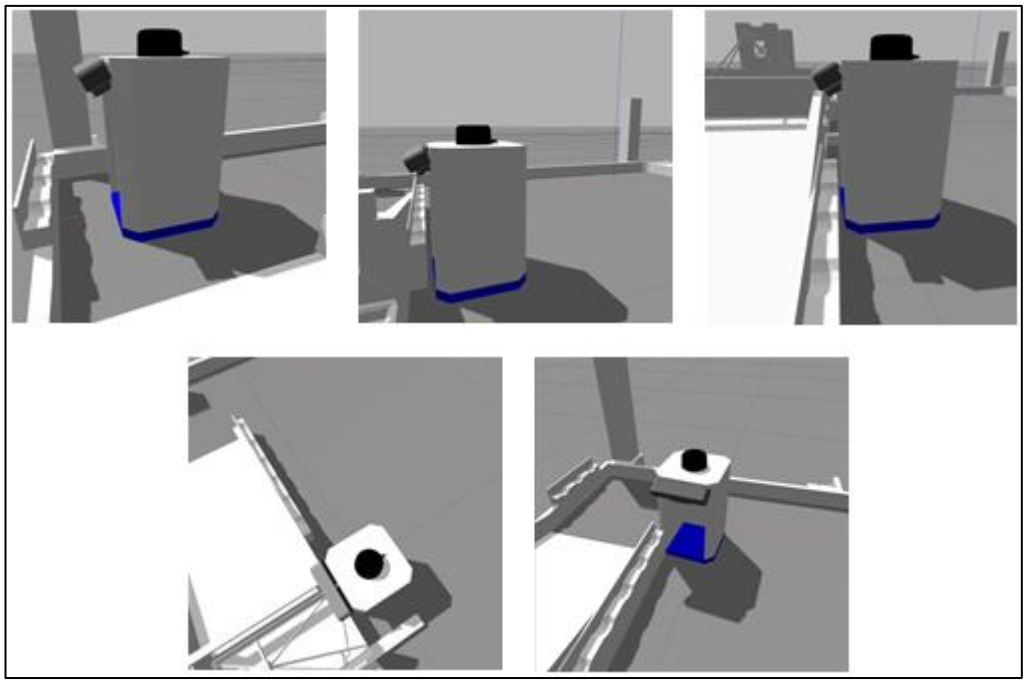

Figura 7.28 De d'alt a baix i d'esquerra a dreta les posicions, en la simulació de Gazebo, dels punts P7, P8, P9, P12 i PA1 després de la primera correcció, és a dir en la segona prova. (Font pròpia)

El fet de considerar els 95,25 mm ha sigut correcte però massa escàs, observant el resultat de la primera correcció (Figura 7.28) P8 i P9 ara estan massa a prop de l'objectiu de tal manera que el robot no podria maniobrar. Per assegurar s'aplica una nova correcció per aquests punts de 100 mm respecte al valor original.

Pel que fa a PA1 tot i la correcció inicial es manté el mateix resultat. Hom recau en el fet que aquest punt d'accés hauria de tenir un valor en l'eix de les x similar o igual al de P8 o P9 per a que no xoqui amb l'estructura, per tant es torna a disminuir el valor en l'eix d'abscisses en la mateixa quantitat (-50 mm) aproximant-se al valor atorgat a P8 sent la diferència de 0,005 m.

P7 tot i estar encara lluny de l'objectiu no es pot considerar disminuir la distància per què d'aquesta manera s'obtindria un resultat com el de P8 o P9, per tant P7 es marca com a correcte i s'ajustarà el que calgui amb el paràmetre d'aproximació del *path planner*.

En tant que P12 el resultat obtingut és del tot incorrecte, en aquest punt s'observa que fins i tot la posició manca d'espai en l'eix de les ordenades i en sobra en el de les abscisses; respecte a la correcció inicial de 100 mm ara se'n fa mitja i s'aplica una correcció de -50 mm en l'eix de les x i de +60 mm en l'eix de les y.

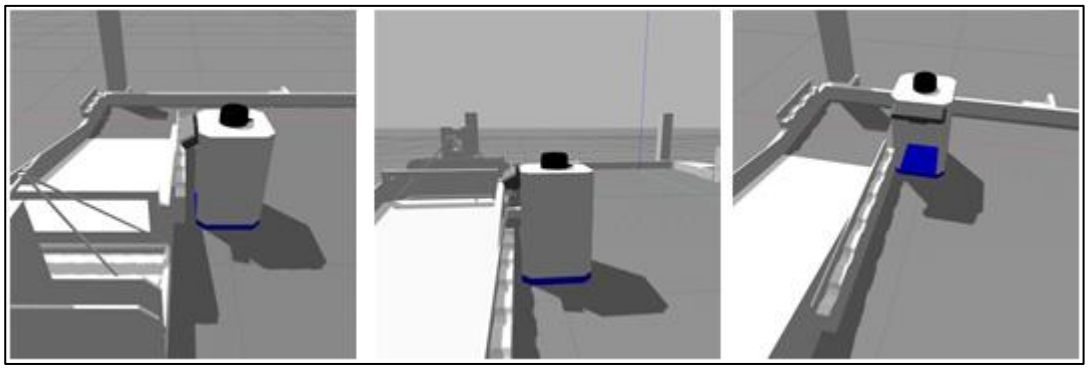

Figura 7.29 Resultats de la segona correcció per a P8, P9 i PA1. (Font pròpia)

El canvi que ha suposat la segona correcció en els punts que es mostren en la figura anterior no ha sigut molt dràstic, tot i així es considera acceptable respecte a lo que s'ha obtingut en la primera correcció. En el cas de PA1, es continua repetint el resultat, es redueix el valor en x per sota del valor en x de P8.

El que no es mostra en la Figura 7.29 és el xoc del robot en el camí des de P8 a P9. Durant la tercera prova ha succeït lo descrit però a l'hora de voler-ho reproduir no s'ha aconseguit. De totes maneres, tot i que la posició del punt es consideri correcte no ho és la forma d'arribar fins a ell, per tant s'ha considerat afegir un pas o *step* en el camí entre RZY i RZP.

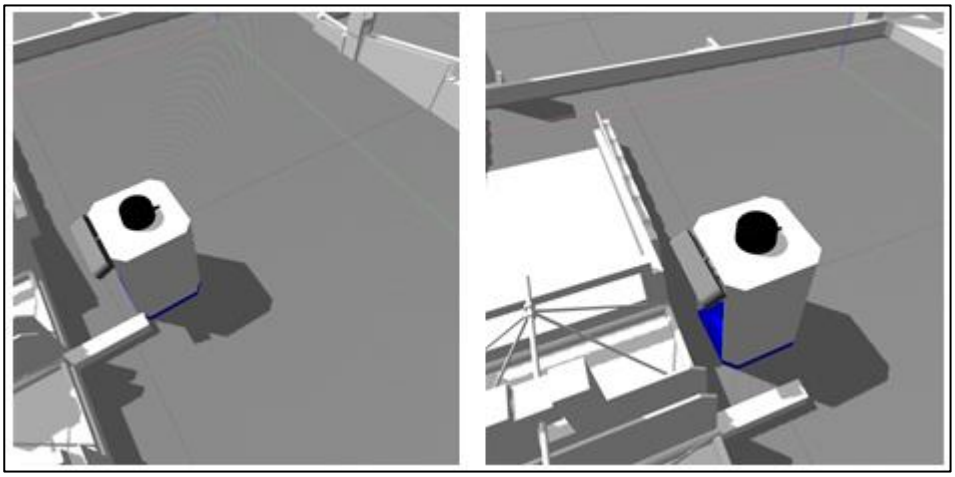

Figura 7.30 Resultats de la segona correcció pel punt P12. A l'esquerra e robot encallat amb l'estructura del camp. A la dreta el robot en el punt P12 venint des de P10. (Font pròpia)

El robot en aquesta tercera prova per accedir a P12 ha patit les conseqüències del problema pel qual s'ha creat l'aplicació. En la figura anterior es veu a l'esquerra el robot encallat amb l'estructura del camp, en l'altre imatge es veu el robot havent arribat al punt però des d'una altre ubicació, per tant també s'hi pot observar l'actuació del paràmetre anteriorment mencionat. Ara bé després d'aquesta prova es pot considerar acceptable el valor en l'eix de les abscisses i es deixa per modificar pel

paràmetre d'aproximació, en canvi el valor en l'eix d'ordenades es sobrepassa. Així doncs fent mitja de les correccions anteriors, en aquest eix s'aplica una altre correcció atorgant-li un valor de 1,32 m.

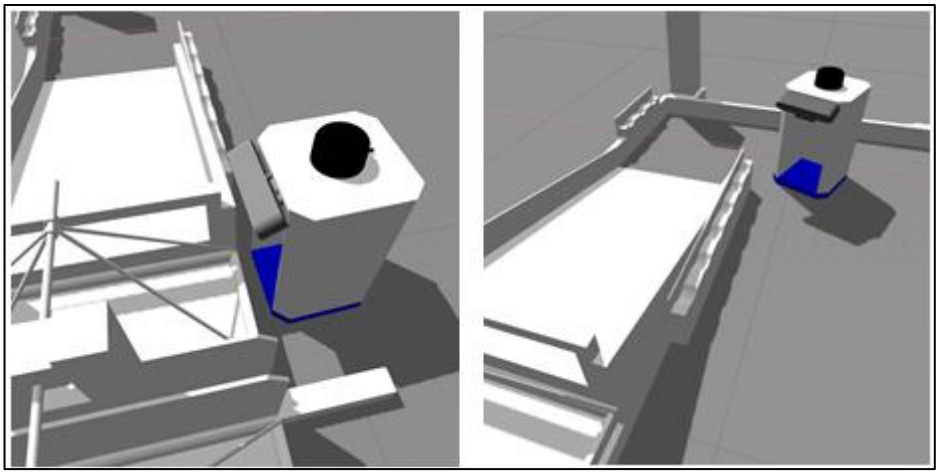

Figura 7.31 Resultats de la tercera correcció pels punts P12 i PA1. (Font pròpia)

Finalment després de tres correccions s'obtenen els resultats representats en la figura anterior, amb un valor en l'eix de les x de 1,35 m a PA1 s'ha aconseguit evitar el xoc amb l'estructura i per tant ja es considera correcte. Pel que fa a P12 tot i tenir mancances en l'eix d'ordenades és més que acceptable donat que encara es pot calibrar modificant el paràmetre repetidament mencionat, per tant P12 també es considera com ha acceptat.

Arribats a aquest punt cal aplicar l'ajust necessari amb el paràmetre d'aproximació. La configuració inicial és la reflectida en la primera configuració de la taula Taula 7.11. Les proves que s'han executat per arribar al valor desitjat tracten d'ajustar la posició d'arribada del robot lo més fidel possible a la marcada; per comprovar-ho es fixa un únic punt per totes les proves, concretament P8, en el qual s'estudia en diferents intents si el robot arriba de la mateixa manera que en el intent anterior, si ho fa vol dir que l'ajust és el correcte.

	config1	config2	config3
xy_goal_tolerance	0,1	0,01	0,05
yaw_goal_tolerance	0,05	0,005	0,05
regulació en el lloc	X	O	X

Taula 7.11 Configuracions de les diferents proves realitzades. (Font pròpia)

En la taula anterior:

- Regulació en el lloc. En diversos proves es dona el cas que el robot a l'hora d'arribar al punt de destí no ho fa de cop. En aquest casos quan el robot s'aproxima al destí, comença un procés de regulació que dura poc temps en el que el robot es mou en totes direccions intentant trobar el punt de destí. Aquest procés es deu a un sobre impuls de la regulació de l'algoritme *path planner* local.
- O. Significa que es dona el cas del sobre impuls que dona peu a la regulació mencionada, es considera un cas no apte.
- X. Significa que no es dona el cas del sobre impuls, es considera un cas apte.

La primera configuració és la que es mostra en les figures anteriors en el procés de correccions, per això es parteix des de la segona configuració.

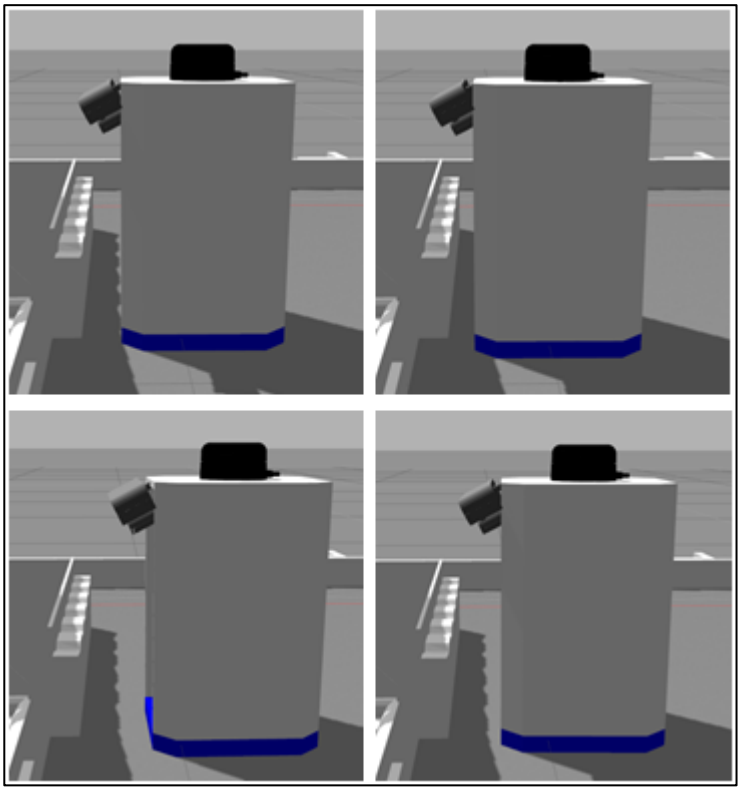

Figura 7.32 Proves amb la segona configuració. (Font pròpia)

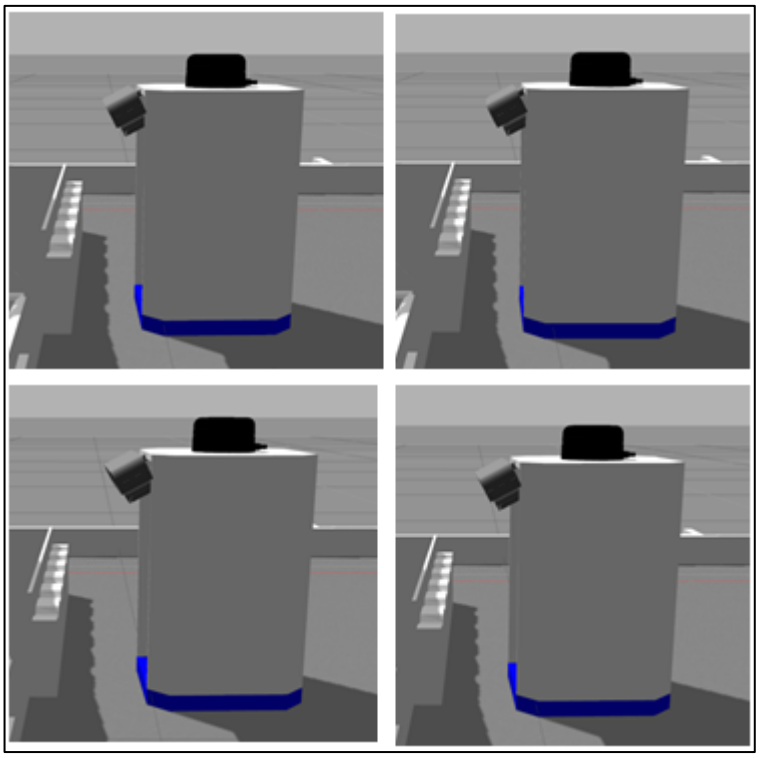

Figura 7.33 Proves amb la tercera configuració. (Font pròpia)

Observant les figures anteriors no es nota gaire la diferència entre una configuració i l'altre. Inicialment degut als inconvenients en les correccions es redueixen ambdós paràmetres en un 90%, els resultats són els que es mostren en la Figura 7.32 i s'observa que en quatre proves entre anar i tornar de P0 el robot és prou fidel a mantenir-se en la posició de P8 sense causar gaire error, ara bé durant les proves s'han registrat regulacions que es deuen a un sobre impuls. Per altre banda, en quant a la tolerància de l'eix de gir no es registren mals posicionaments en aquest sentit abans de la segona configuració.

És per tot que es proposa una tercera configuració, retornant al valor original el paràmetre de tolerància de l'eix z, mentre que pels eixos x i y s'augmenta el valor +0,04 quedant en 0,05. Amb aquest valors, els resultats són els que es mostren en la Figura 7.33, res més lluny del que s'ha vist anteriorment amb la segona configuració, tot i així en quatre proves no es registren casos de cap regulació; per tant aquesta és la configuració final.

8. Propostes de millora

En aquest apartat es fa balanç del que s'ha portat a terme en el projecte, indicant de manera breu quines millores es podrien adoptar al projecte en:

8.1. Arduino

Ja que el robot en la seva versió final incorporava Arduino una millora podria basar-se en aquest aspecte. Més concretament, ROS incorpora un *package* capaç de crear una comunicació amb aquest tipus de hardware. De tal manera que com a punt de partida abans de implementar altres alternatives de més rendiment, s'implantaria l'opció d'Arduino per fer moure el robot seguint les indicacions de les simulacions des de ROS.

És a dir, arribant al punt d'aplicar una estratègia amb l'aplicació de python creada, amb un altre node de tipus *Publisher* es podria comunicar via port sèrie els moviments a fer. El tòpic *cmd_vel* és l'encarregat de transportar la informació de la velocitat i direcció del robot, de la mateixa manera que *odom* però de forma més reduïda. Publicant aquest tòpic, des d'Arduino s'hauria d'interpretar el missatge, desglossar-lo, traduir-lo amb un node *Subscriber* i aplicar tal missatge als divers dels motors per tal de moure les rodes.

D'aquesta manera es passaria d'una simple simulació per ordinador a una de sincronitzada a temps real amb el robot físic que era un dels objectius del plantejament inicial del projecte. Val a dir que ja s'han realitzat proves satisfactòries en torn a aquesta idea, tot i que utilitzant l'aplicació per poder moure el robot virtual via teclat.

8.2. Utilització d'altre hardware

Amb aquest apartat hom es refereix al hardware que per exemple el robot original o el *package* original incorporen i que no s'han fet servir. No obstant l'abast del projecte limita certs aspectes, per exemple pel que fa a la càmera de profunditat, era part d'un altre treball dins del projecte portat a terme per altres companys; amb aquesta es pretenia "ajudar" al LIDAR a escanejar el terreny, aportant dades de posicions dels objectes en el camp. Aquest treball s'inicia amb l'intent de reconeixement dels discs mitjançant visió per computador amb OpenCV. Ara bé de la mateixa manera que amb el treball que hom ha realitzat aquest mencionat va tenir dificultats per avançar i

per tant va quedar inacabat. Tot i així, diferents *packages* de ROS – com l'anomenat Aruco - incorporen la funcionalitat de reconeixement d'objectes en simulació de tal manera que en un futur projecte es podria incorporar el reconeixement dels discs en les simulacions.

Altre hardware que es podria utilitzar són els sensors de proximitat i les pinces. Ja el *package* original incorporava els sensors d'infraroig en el seu robot, de totes maneres no hi havia cap *plugin* en aquest *package* que li donés ús, així doncs es proposaria incorpora-ho al robot final i donar-li l'ús que es requereix. Pel que fa a les pinces, gràcies al *package* MoveIt!, ROS incorpora més funcionalitats per treballar amb braços robòtics.

8.3. Aplicació python

Aquesta aplicació ha servit de gran ajut per finalitzar el treball, donant un punt de recolzament al càlcul de trajectòries. Un dels aspectes a millorar seria: la incorporació de la resta de punts de interès en el camp per tal de poder realitzar les estratègies amb el equip morat; millorar la fluència del codi fent possible poder sortir de l'aplicació en execució; i incorporar l'opció d'afegir enemics des de la mateixa aplicació indicant la posició. Amb tot això es consideraria l'aplicació com a finalitzada.

8.4. Altres

Pel que fa al *Machine Learning* que es proposava realitzar des d'un principi, podria fer-se aplicant a cada punt d'interès del mapa un cost de puntuació i de temps, amb aquestes dues variables i comptant amb la variable de distància a recorre des del punt inicial fins al suposat destí, es podria crear un algoritme capaç de discriminar un destí o un altre. Mitjançant aquest càlcul el robot hauria de trobar-se amb centenars de possibilitats i executar-ne l'algoritme per a cada una de manera que aplicant l'aprenentatge seria possible crear una màquina mínimament intel·ligent. Aquesta però és només una suposició de la qual no se'n té base total de coneixement per afirmar-la ni portar-la a terme, de totes maneres hom creu possible l'opció de desenvolupar-la i realitzar-ne sí més no una idea similar.

Per altre banda, una de les limitacions importants del projecte és el fet d'haver escanejat un espai del qual no se n'ha pogut reconèixer els obstacles d'aquest, això degut a la normativa de la competició com ja s'ha comentat anteriorment. De totes maneres, seria més útil realitzar la idea anterior amb un altre tipus d'escanejat. Una de les opcions seria realitzar el mateix procés reduint la coordenada z del

sensor LIDAR, fins al punt que aquest pogués visualitzar aquests obstacles mencionats. L'altre opció recau en utilitzar la torre de *tracking* del mateix camp, preferiblement amb una càmera per tal de poder identificar els robots amics i els enemics via visió per computador.

Una altre solució que es podria adoptar vers a aquest inconvenient és l'ús dels *markers*, amb els que es podrien definir les zones del camp quedant representades en l'entorn gràfic de Rviz. No afectaria al càlcul de trajectòries ja que aquest només treballa amb els valors que proporciona el camp escanejat i el LIDAR, de totes maneres fent uns *markers* interactius es podria aconseguir evitar que el robot s'encallés en certs casos que no es pogués evitar el xoc.

9. Anàlisi de l'impacte ambiental

El treball que es presenta no comporta un impacte directe en el medi ambient. No obstant filant prim, en cas de portar a més aquest projecte es deriva en la utilització de material electrònic i mecànic per a la construcció del robot. El qual en un principi, entre altres coses, incorporava cel·les de Liti creant així una bateria. Per tant l'ús d'aquest material, si no es fa de la manera correcte pot causar un impacte en el medi ambient.

Des de l'associació la reutilització del material és clau. A l'hora d'acabar una competició, es classifica tot element que pugui ser útil però el que no, es tracte de reciclar en la mesura del possible. En la següent taula més avall es mostra, en cas de generar residus, quin és el procediment segons el seu tipus.

Tal com estableix la Llei reguladora dels residus 6/1993 del govern català, basada en la directiva europea 91/689/CE, els residus es classifiquen segons la seva composició per la seva perillositat de la següent manera:

- Residu Inert: Es considera residu inert aquell que, un cop disposat en un abocador, no experimenta cap transformació física, química o biològica significativa, i a més compleix els criteris de lixiviació determinats per reglament.
- Residu especial: És residu especial tot residu comprès en l'àmbit d'aplicació de la Directiva 91/689/CE, del 12 de desembre.
- Residu no especial: És residu no especial tot residu no classificat com a residu especial o com a inert. (Generalitat de Catalunya, 1993)

Tot i que de totes maneres, en aquesta llei es fa una interpretació efímera nombrant residus especials i no especials als anomenats per la Directiva europea com a *hazardous* (perillosos) i *not hazardous* (no perillosos). En aquest sentit en la Taula 9.1 es classifiquen els residus generats en l'associació segons aquest criteri de perillositat. Pel que fa a aquests són de caire No Perillosos la majoria d'ells ja que pertanyen al grup de residus municipals segons s'indica en el Sistema Documental de Residus de l'Agència de residus de Catalunya. (Agencia de Residus de Catalunya, 2016)

TIPUS DE RESIDUS		Descripció	Classe	Reciclatge
Metàl·lics	Fèrrics	Ferro i acer	NP	Punt Verd / Deixalleria
	No Fèrrics	Alumini, coure, estany	NP	Punt Verd / Deixalleria
Fusta		Retalls de fusta, taulons, mobles o peces de mobles	NP	Punt Verd / Deixalleria
Bateries		Cel·les LiPO de Ni-Cd	P	Punt Verd / Deixalleria
Electrònica		Components electrònics, ordinadors, teclats, ratolins i pantalles	P/NP	Punt Verd / Deixalleria
Paper		Retalls de paper, fulls de paper (fulls de càlcul o de notes, plànols, etc.), cartrons	NP	Contenidor Blau
Plàstics		Envasos, bosses de plàstic, plàstic protector de bombolles, brides, termoretràctils	NP	Contenidor Groc
Pintures		Pintures fetes servir com a decoració del robot o dels camps de competició	NP	Punt Verd / Deixalleria

Taula 9.1 Taula de classificació de residus que es generen en el taller de l'associació en la fabricació d'un robot. (Font pròpia)

Conclusions

Per una banda es tanca aquest treball sent hom conscient de la utilitat de ROS en tantes aplicacions: havent només rascat la superfície de possibilitats que aquest *framework* ofereix s'ha pogut recrear un *package* capaç de simular el moviment d'un robot en un espai físic, tant de manera controlada com autònoma. Durant el projecte inicial es va poder constatar la importància en reposar-se de canvis sobtats, de manera que un sigui capaç de tirar endavant vers les dificultats.

Per altre banda, l'aplicació creada, és a dir: l'escanejat d'espais simulats donant peu a poder recrear estratègies dins d'un camp de competició; ha servit com a punt de partida per a futurs projectes o millores d'aquest. Tot i certs inconvenients durant el procés, és clar que per aplicacions de baix rendiment com aquesta, Dijkstra és l'algoritme òptim per trobar el camí més curt. No obstant, existeixen desenes d'algoritmes modificats basats en el del científic en computació (entre aquests A*), els quals podrien haver-se inclòs en aquest projecte; ara bé aquesta conclusió ve determinada per l'abast del treball, és a dir les opcions que ROS ofereix per defecte.

I seguint en aquesta línia, *dwa_local_planner* és un bon algoritme que atorga la major velocitat possible al moviment del robot per tant de molta utilitat si es tracte d'estalviar temps, en canvi, *eband_local_planner* estableix més seguretat en els moviments assegurant el mínim de xocs possibles. Doncs per una banda, en tant que a la millor opció per la competició es refereix: l'algoritme de DWA és òptim per aportar el màxim rendiment al robot ja que el temps és limitat per actuar en les partides, en canvi la màxima velocitat pot ser una insegura aposta per evitar xocs amb els contrincants i per tant la pèrdua de punts, ara bé que un millor calibratge pot haver deixat que desitjar d'aquest algoritme; per altre banda l'algoritme de la banda elàstica, tot i aparentar ser un *path_planner* més òptim com a global, la seva funció com a local dona la seguretat d'evitar el màxim de xocs possibles amb la configuració de les seves bombolles. Si més no, el fet d'incorporar un robot en el projecte amb moviment holonòmic deixa moltes opcions fora de lloc, en definitiva la combinació que s'ha configurat en aquest projecte és creu l'òptima.

En definitiva s'ha trobat una solució pel que fa a la mobilitat autònoma del robot en tant que a simulacions es refereix. Partint d'aquest treball es poden plantejar millores i addicions de noves idees, fins al punt d'implementar el software configurat al robot físic de tal manera que es puguin aconseguir els mateixos resultats.

Anàlisi Econòmica

Per aquest apartat es representa el projecte en petit estudi de negoci, *business case*. S'ha realitzat un resum d'inversió i costos del primer any, mentre que per 3 anys vista s'han valorat les possibles correccions futures del software. Val a dir que aquest projecte esta purament centrat en software i doncs per aquest motiu els costos són simples. La inversió inicial consta de la compra d'un ordinador per part de l'associació com a requeriment bàsic donades les condicions del projecte. Més endavant es fa una segona inversió, un segon ordinador d'ús personal que compleix amb els requisits per seguir endavant el projecte. Es comptabilitzen les hores produïdes de manera aproximada per a la confecció de tot el software, imposant un preu hora just segons la condició de l'autor. A 3 anys vista es compta amb la millora del software suposant un 20% del cost de les hores produïdes el primer any.

		hores (mitjana setmanal)	setmanes	€/h	Total Mes
2018	Setembre	6	4	12	288
	Octubre	5	4	12	240
	Novembre	5	4	12	240
	Desembre	3	3	12	108
2019	Gener	20	2	12	480
	Febrer	20	4	12	960
	Març	15	4	12	720
	Abril	15	4	12	720
	Maig	25	4	12	1200
	Juny	17,5	4	12	840
	Juliol	17,5	4	12	840
	Agost	17,5	4	12	840
Total		166,5	45	-	7476

Taula 0.1 Desglossament del càlcul aproximat pel cost de desenvolupament. (Font pròpia)

		2019	2020	2021	2022
Inversió		1200	0	0	0
Correccions		0	1495,2	1495,2	1495,2
Costos	Llibres	25	0	0	0
	Desenvolupament	7476	0	0	0
	Total costos	7501			

Taula 0.2 Resum del *business case*. (Font pròpia)

Bibliografia

Agencia de Residus de Catalunya (2016) *SDR - Sistema Documental de Residus*. Disponible a: https://sdr.arc.cat/modemp/ConsGestor.do?codiGestor=E-1497.14&posInicial=13 (Accedit: 27 agost 2019).

Dellaert, F. *et al.* (1999) «Monte Carlo Localization for Mobile Robots», *Icra*, 2, p. 1322-1328. Disponible a: https://www.cc.gatech.edu/~dellaert/ftp/Dellaert99icra.pdf%0Ahttp://ac.els-cdn.com/S0004370201000698/1-s2.0-S0004370201000698-main.pdf?_tid=ee7a0eac-c419-11e3-ae22-00000aab0f02&acdnat=1397510320_c8498bb43627ba8e16db14ba87856cb0.

Djikstra, E. W. (1959) «A Note on Two Problems in Connexion with Graphs», *Numerische Mathematik*, 1(1), p. 269-271.

Eurobot (2019a) *eurobot.org*. Disponible a: http://www.eurobot.org/ (Accedit: 30 juliol 2019).

Eurobot (2019b) «Eurobot Rules 2019». Disponible a: http://www.eurobot.org/images/2019/Eurobot2019_Rules_Cup_OFFICIAL_EN.pdf.

Fox, D., Burgad, W. i Thrun, S. (1997) «The Dynamic Window Approach to Collision Avoidance», *IEEE Robotics & Automation Magazine*, 4(1), p. 137-146. Disponible a: https://www.ri.cmu.edu/pub_files/pub1/fox_dieter_1997_1/fox_dieter_1997_1.pdf.

Generalitat de Catalunya (1993) *Llei reguladora dels residus*. Espanya. Disponible a: http://www.arc.cat/ca/publicacions/pdf/normativa/catalana/lleis/llei_6_1993.pdf.

GitHub (2007) *GitHub*. Disponible a: https://github.com/ (Accedit: 16 febrer 2019).

Open Source Robotic Fundation (2011) *ROS wiki*. Disponible a: http://wiki.ros.org/ (Accedit: 5 setembre 2018).

Open Source Robotics Foundation (2014) *Gazebo*. Disponible a: http://gazebosim.org/ (Accedit: 20 setembre 2018).

Peter, E. H. (IEEE), Nils, J. N. (IEEE) i Raphael, B. (1968) «A Formal Basis for the Heuristic Determination of Minimum Cost Path», *IEEE Transactions of Systems Science and Cybernetics*, 4(2), p. 100-107.

Quigley, M. *et al.* (2010) «ROS : an open-source Robot Operating System». Disponible a: https://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf.

Quinlan, S. i Khatib, O. (1993) «Elastic Bands: Connecting Path Planning and Control», *IEEE International Conference on Robotics and Automation*. Disponible a: http://www8.cs.umu.se/research/ifor/dl/Control/elastic bands.pdf.

Téllez, R., Ezquerro, A. i Rodríguez, M. A. (2018) «ROS Navigation in 5 days». Barcelona: The Construct. Disponible a: www.theconstructsim.com.

Slamtec (2013) «RPLIDAR A2 datasheet». Disponible a: https://www.robotshop.com/media/files/pdf2/ld208_slamtec_rplidar_datasheet_a2m8_v1.0_en.pdf.